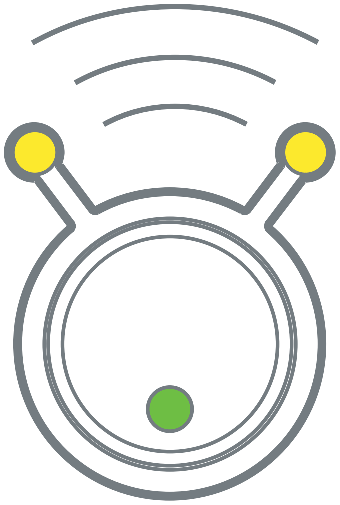
The Playful Invention Company



ピコクリケット
取扱説明書

Version 1.2a

謝辞

- デザインチーム
  - Mitchel Resnick
  - Brian Silverman
  - Paula Bontá
  - Robbie Berg
  - Natalie Rusk
- インダストリアルデザイン：Smart Design
- グラフィックデザイン：Big Blue Dot

ピコクリケットの開発に協力してくださった次の方々に感謝いたします

- 貴重なアイデアを提供し、情熱を注いでくれたPICO社スタッフのChad Burt、Catherine Cournoyer、Mike Durcak、Jack Geddes、Laurelle MiciakそしてMari Moreshead
- オンラインヘルプ担当のDiana Gee Silverman、サウンド担当のDanny Lutz、コンプライアンス試験および製造設計という複雑な側面でPICO社を指導してくれたLawrence Shubert
- クリケットの新機能を開発するPIE（Playful Invention and Exploration）のスタッフ、特にKeith Braafladt、Stephanie Hunt、Chip Lindsey、Hideki Mori、Kristen Murray、Mike Petrich、Margaret Pezalla-Granlund、Natalie Rusk、Michael Smith-Welch、Karen Wilkinson、Diane Willow
- クリケットの研究開発に貢献してくださったMITの研究者と学生の皆さん、特にAndy Begel、Robbie Berg、Rahul Bhargava、Rick Borovoy、Fred Martin、Bakhtiar Mikhak、Mitchel Resnick、Brian Silverman
- 変わらないご支援をくださったLEGO 社の皆さん、特にKjeld Kirk Kristiansen、Lisbeth Valther Pallesen、Jens Maibom、Erik Hansen

© 2006-2007. The Playful Invention Company Inc. 無断複写・転載を禁じます
本書は、著作権所有者の書面による事前承認を得ることなく、複製、表示、変更または配布できません

目次

謝辞	**2**
はじめに	**4**
ピコブロックス	**8**
ブロックの概要	**12**
ブロックーカテゴリ別の詳細	**14**
メロディエディタとリズムエディタ	**32**
データ表示	**37**
ピコブロックステキスト言語	**38**
一般的なエラーメッセージ	**52**

はじめに

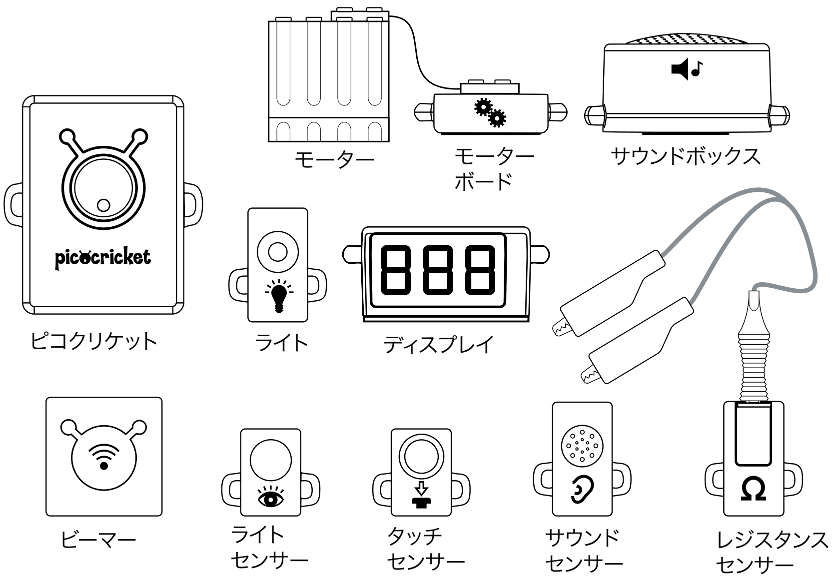
この取扱説明書では、ピコクリケットファミリーのパーツとピコクリケットをプログラムする際に使用するピコブロックスソフトウェアの使い方について説明します。

はじめてお使いになる際には、まず別冊の『セットアップガイド』と『やってみよう！』をお読みいただきますと概要がご理解いただけます。
さらに詳しい情報が必要な場合には、この『取扱説明書』をお読みください。

弊社では『Reference Guide（取扱説明書）』を随時更新しています。
最新版はwww.picocricket.com/supportからダウンロードできます。

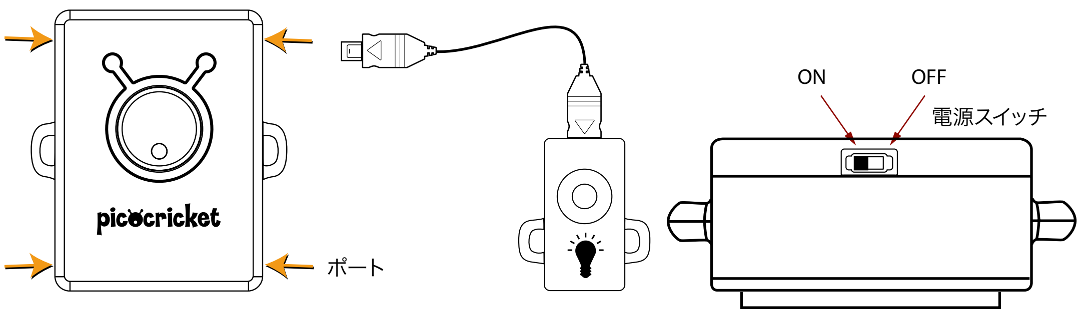
ピコクリケットファミリー

ピコクリケットファミリーには以下の機器が含まれています。

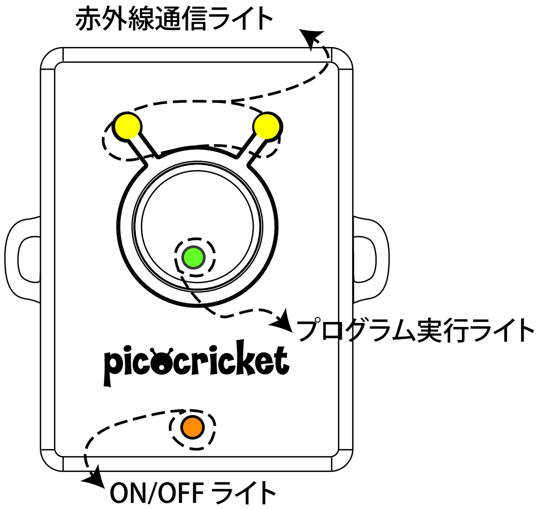


ピコクリケット

ピコクリケットは、いろいろなものをコントロールできる小さなコンピュータです。
ピコクリケットには、センサー、モーター、ライトなどの機器を接続できる4つのポートがついています。
これらの機器は、どのポートにでも接続することができます。



ピコクリケットには、
3種類のインジケーターライト(赤外線通信ライト、プログラム実行ライト、ON/OFFライト)があります

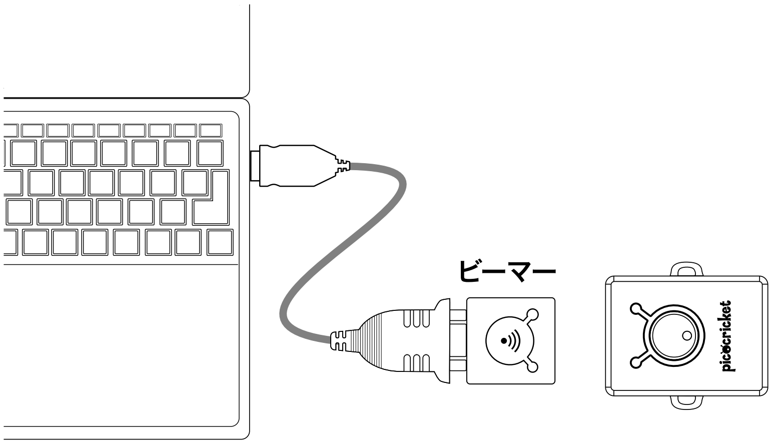


- ピコクリケットの電源スイッチを入れると、ON/OFF ライト(オレンジ色)が点灯します。電池が消耗すると、ON/OFFライトが点滅します。
- ピコクリケットでプログラムを実行している間は、プログラム実行ライト(緑色)が点灯します。また同時に2つのプログラムを実行している時は、濃い黄色に変わります。
- ピコクリケットがビーマー(赤外線送信機)で通信している間は、赤外線通信ライト(黄色)が点灯します。

プログラムを実行している時にピコクリケットのボタンを押すと、プログラムが停止します。
プログラムを実行していない時にボタンを押すと、最後に実行したプログラムを再実行します。

ピコクリケットの電源を切っても、最後に実行したプログラムは記憶されています。
ピコクリケットの電源を入れ、ボタンを押すと、ピコクリケットは最後に実行したプログラムを再び最初から実行します。

ビーマー



ビーマーは、コンピュータから赤外線信号でプログラムをピコクリケットに送信する装置です。
コンピュータでピコブロックスソフトウェアを使って作成したプログラムは、ビーマーを経由してピコクリケットに送信されます。

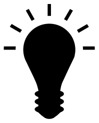
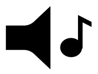
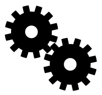
ピコクリケットキットには、コンピュータのUSBポートにビーマーを接続するためのUSBシリアルケーブルが付属しています。また、シリアルポートのあるコンピュータをお使いの場合は、別売りのシリアルケーブルを使用して、ビーマーを直接シリアルポートに接続することもできます。

プログラムをピコクリケットに送信する時は、ビーマーのアンテナとピコクリケットのアンテナが向き合っていることを確認してください。ビーマーとクリケットが通信している間は、ビーマーとピコクリケットの赤外線通信ライトが点滅します。

ビーマーの通信範囲は約1メートルです。ビーマーが一度に通信できるのは1台のピコクリケットのみです。複数のピコクリケットを使用している場合は、ビーマーの通信範囲内にあるピコクリケットが常に1台のみになるようにしてください。

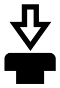
アクチュエータ

アクチュエータは、さまざまに動作する出力機器です。ピコクリケットキットには、ライト、サウンドボックス、モーター(およびモーターボード)とディスプレイの4種類のアクチュエータが付属しています。

名称	用途	ブロックの種類	アイコン
ライト	ライトの色を赤から紫まで、さまざまな色に設定できます。 ライトの明るさも設定できます。	setlight color / setlight power / light off	
サウンドボックス	サウンドボックスは、単音、メロディ、効果音およびリズムを演奏できる小さなシンセサイザーです。 ピコブロックスソフトウェアには、メロディとリズムを作成するエディタがあります。 どの楽器を使って音やメロディを演奏するか、設定や変更をすることもできます。	play sound / set inst / set volume / set tempo / note	
モーターおよびモーターボード	モーターボードとクリケットに接続し、LEGOモーターとモーターボードを接続します。 モーターのON/OFF、回転速度、回転方向を設定することができます。	motor on / motor off / motor onfor / this way / that way / reverse / set power	
ディスプレイ	3桁のディスプレイには、0から999までの数値を表示できます。 特に、センサーの値を表示する時に役立ちます。	display	888

センサー

センサーを使用すると、ピコクリケットは周囲の音や光、その他の変化に反応することができます。

名称	用途	ブロックの種類	アイコン
ライト センサー	光の明るさを調べます。 たとえば、ライトセンサーを使用すると、誰かの手で影ができたり、太陽の光が窓から差し込んだことを感知することができます。	dark? brightness	
サウンド センサー	音声の大きさ（ボリューム）を調べます。 たとえば、サウンドセンサーを使用すると、誰かが手をたたいたり、誰かがセンサーに向かって音を立てたことを感知することができます。	loud? loudness	
タッチ センサー	ボタンを押したかどうかを調べます。 たとえば、誰かがタッチセンサーを押すと、何らかの機能（モーターやライト）が作動するようにプログラムを組むことができます。	touch?	
レジス タンス センサー	ワニ口クリップ付きケーブルを使って抵抗を調べます。 ワニ口クリップの間にさまざまな素材（粘土、バナナなど）を置くと抵抗が変わります。また、レジスタンスセンサーを使用すると、ワニ口クリップが互いに接触していること（または、アルミホイルなどの導電体を介してつながっていること）を感知することができます。	connected? resistance	Ω

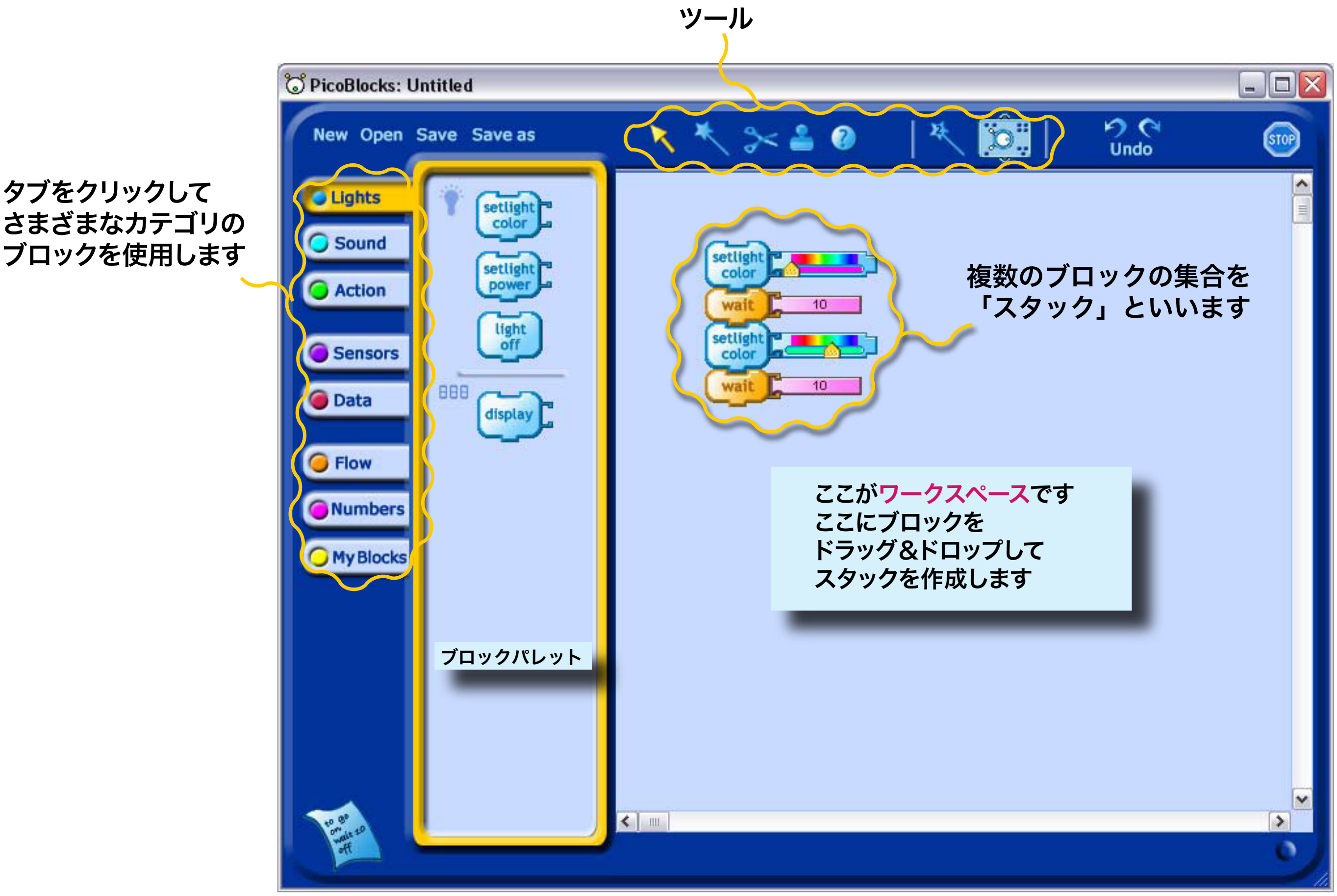
各ブロックに関しては次ページ以降で詳しく説明します。
アクチュエータとセンサーの使い方については、「魔法のランプ」などのプロジェクトプレースマットも参照してください。

クリケット取り扱いに関する注意

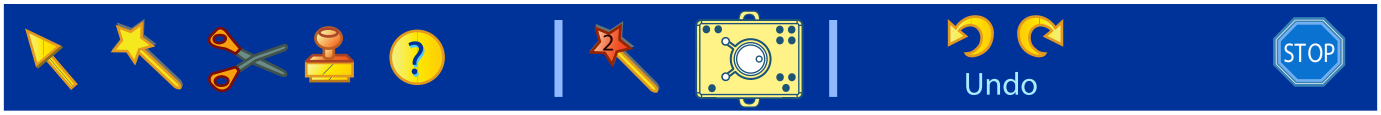
1. 使用しないときは、必ずピコクリケットの電源を切ってください。
2. オレンジ色のON/OFFライトが点滅する場合は、クリケットの電池が消耗しています。
   電池交換の際は、『セットアップガイド』に参照してください。
3. ケーブルコネクタは丁寧に扱ってください。
   必要以上に強い力でケーブルコネクタを上下に動かすと壊れる可能性があります。
4. ピコクリケットは水などの液体に近づけないでください。
5. ピコクリケットがうまく動作しない場合は、ウェブサイトのトラブルシューティング
   www.picocricket.com/troubleshooting を参照してください。

概要

ピコブロックスは、ピコクリケットをプログラムするために作られたプログラミング言語です。
ピコブロックスでは、グラフィカルなブロックをスタックにはめ込むことでプログラムを作成することができます。



ツール



ツールはブロックのコピーや削除、ピコクリケットへの送信など、ブロックのさまざまな処理を設定する際に使用します。ツールをクリックすると、カーソルがそのツールに変わります。

矢印

矢印はブロックパレットからワークスペースに、またはワークスペース内でブロックをドラッグする時に使用します。スタック全体を移動する時は、先頭のブロックをドラッグします。
スタックの中間からブロックをドラッグすると、ドラッグするブロックとその下に続くブロックを移動することができます。ブロックを削除する場合はブロックをワークスペースの外にドラッグします。
誤ってブロックを削除してしまった場合は、[取消し] をクリックします。

魔法のつえ

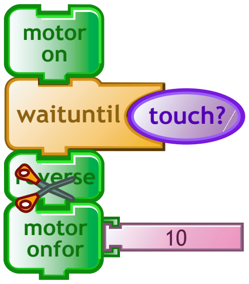
[魔法のつえ]はプログラムを実行するために使用します。
[魔法のつえ]でブロックまたはスタックをクリックすると、ピコクリケットにプログラムを送信して、プログラムを実行するようにピコクリケットに指示します。

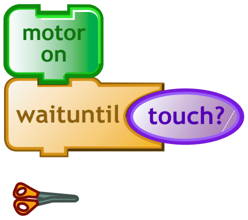
[魔法のつえ]は、ワークスペース上だけではなく、ブロックパレット上でも使うこともできます。
この機能はブロックをテストする時に役立ちます。
また、[矢印]でダブルクリックすることでも、プログラムを実行することができます。

はさみ

[はさみ]はブロックを取り除くために使用します。
[はさみ]でスタックの中のブロックをクリックすると、その下に続くすべてのブロックが削除されます。
誤ってブロックを削除した場合は、[取消し] をクリックします。

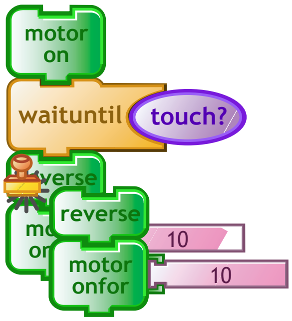
ブロックをワークスペースの外までドラッグすることで、ブロックを削除することもできます。





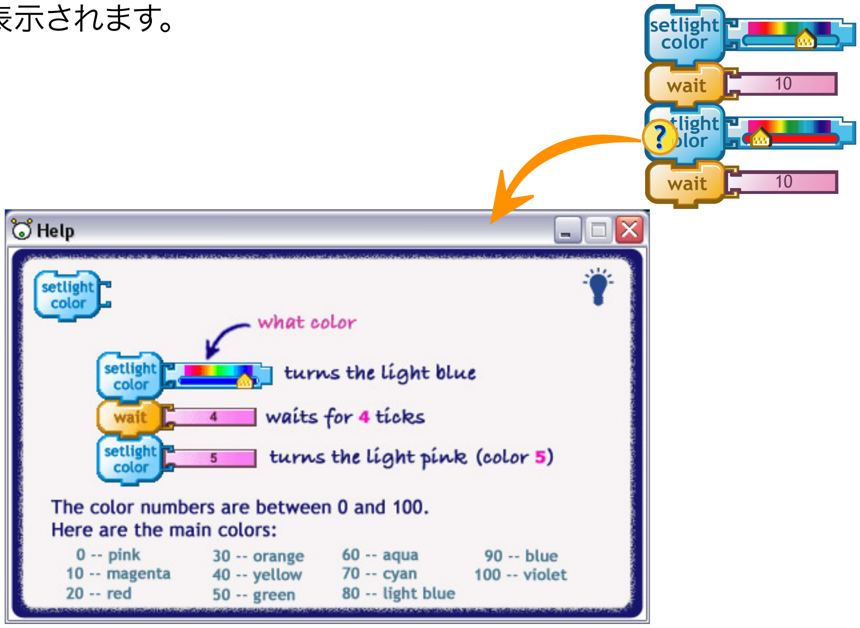
スタンパー

[スタンパー]はブロックやスタックをコピーする時に使用します。一度クリックするとコピーし、
別の場所でもう一度クリックするとコピーしたブロックやスタックが貼り付けられます。
[スタンパー]でスタックの中にあるブロックをクリックすると、下に続くすべてのブロックがコピーされます。
Ctrlキー＋ または キー＋ を押すと、一時的に[スタンパー]に切り替えることができます。



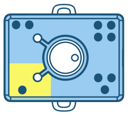
ヘルプ

[ヘルプ]はより詳しい情報を表示したい時に使用します。
[ヘルプ]ツールを選んで、ブロックをクリックすると、別のウィンドウにそのブロックに関する詳しい情報が表示されます。



第2のつえ

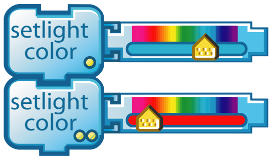
[第2のつえ]は、同時に2つのスタックを実行する時に使用します。
2つのスタックを実行する時は、[魔法のつえ]で1つ目のスタックをクリックし、[第2のつえ]で2つ目のスタックをクリックします。
たとえば、ピコクリケットがあるプログラムを実行している間に、[第2のつえ]で2つ目のスタックを送信することで、1つ目のプログラムを実行しながら、音楽をスタートさせたり、センサーの値を表示させることができます。



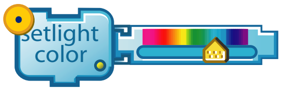
タグ

[タグ]ツールは、2つの同じタイプの機器に別のことを実行させる時に使用します。
たとえば、ピコクリケットに2つのライトを接続して、setlightcolorコマンドを送信すると、両方のライトの色が変わります。もし、各ライトをそれぞれ別の色にしたい時は[タグ]を使用します。

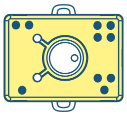
ピコクリケットの4つのポートには、異なる数のドットマークが付いています。
特定のポートにコマンドを送信する場合は、ブロックにそのポートと同じ数のドット「タグ」を付ける必要があります。たとえば、ポート ● と ●● にカラーライトを接続している場合、次のブロックで各ライトの色を設定することができます。

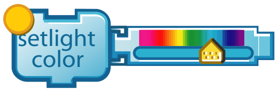


ブロックにタグを付けるには、[タグ]ツールでドットを選び、タグを付けるブロックをクリックします。



ブロックからタグを取り除くには、[タグ]ツールの中央をクリックし、ブロックをクリックします。





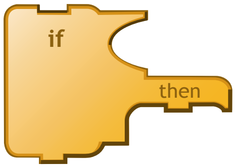
ストップ

[STOP]をクリックすると、ピコクリケットが実行しているすべてのプログラムを停止させることができます。ピコクリケットへ[STOP]命令を送るために、ピコクリケットがビーマーの「近く」にあることを確認してください。ピコクリケットの白いボタンを押すことでも、プログラムを停止させることができます。
ピコクリケットのプログラムを停止しても、クリケットに接続されているライトは消えません。ライトを消す場合は、電源スイッチをOFFの位置までスライドさせてください。

ピコブロックス基本ブロック一覧

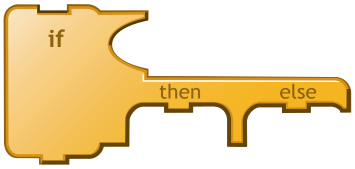
ブロック	説明
setlight color	ライトの色を設定
setlight power	ライトの明るさを設定
light off	ライトを消す
display	ディスプレイに数値を表示
chirp	「チッ」と鳴らす
note	音を鳴らす (音の高さと長さを設定)
play sound (Kitten)	サウンドを再生
set inst (Piano)	楽器を設定
set tempo / set volume	テンポ 音量を設定
motor on / motor off	モーターをON OFF
motor onfor	モーターを指定時間ONにする
reverse	モーターの回転方向を反転
this way / that way	モーターの回転方向を設定
set power	モーターの強さを設定 (モーターの強さと回転速度を変える)
touch?	タッチセンサーが押されているか判別する
brightness	ライトセンサーの値
dark?	ライトセンサーが暗さを感知しているか判別する
loud?	サウンドセンサーが大きな音を感知しているか判別する
loudness	サウンドセンサーの値
connected?	レジスタンスセンサーが抵抗0を検出しているか判別する
resistance	レジスタンスセンサーの値
beamIR	別のピコクリケットに赤外線でデータを送信
IR	別のピコクリケットから赤外線で送信された値
newIR?	別のピコクリケットが赤外線で新しい値を発信しているか判別する
timer	ピコクリケットのタイマー値(1/100秒単位)
reset timer	ピコクリケットのタイマー値を0に設定
clear	データ全消去
collect	ピコクリケットのメモリに数値を記憶する
rewind	始めのデータに戻る
next data	ピコクリケットに記憶されている次のデータの値
wait	指定したticksの間、待つ (1 tick = 1/10秒 ： 10 ticks = 1秒)
waituntil	指定された条件を満たすまで待つ
forever	ブロックまたはスタックを繰り返す
repeat	ブロックまたはスタックを指定した回数だけ繰り返す

ブロックの概要（続き）



指定した条件が満たされた場合、
ブロックまたはスタック(*)を実行

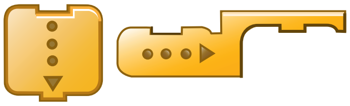
＊スタック：複数から成るブロックの集合を指します



指定した条件が満たされた場合、
左側のスタックを実行；
満たされない場合、
右側のスタックを実行

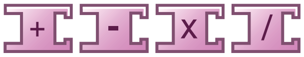
すべてを停止

このスタックを停止



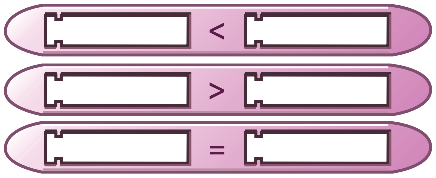
ブロック同士が重ならないように、
ブロックの 間にスペースを作成

数値を指定（クリックして強調表示し、数値を入力）

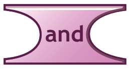


四則計算(＋／−／×／÷)

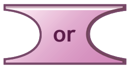
2つの数値の間で乱数を発生



2つの数値を比較し、条件を満たしているか、
判別する

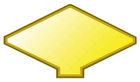


両方の条件を満たしているか、判別する

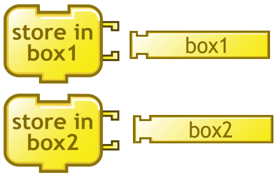


どちらかの条件を満たしているか、判別する

指定した条件の反対（の条件）を満たしているか、判別する

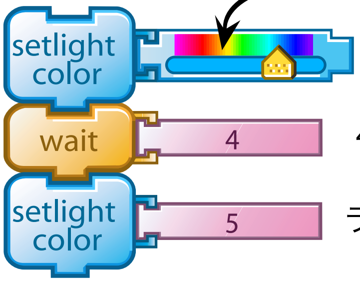


スタックの上に置いて、スタックの名前を入力



変数の値を設定

ライトブロック



ライトを青にする

4 ticks 待つ

ライトをピンク(color 5)にする

ライトの色を設定します。色を設定するには、スライダーを使用するか、0～100の数値を入力します。各色の数値は次のようになっています。

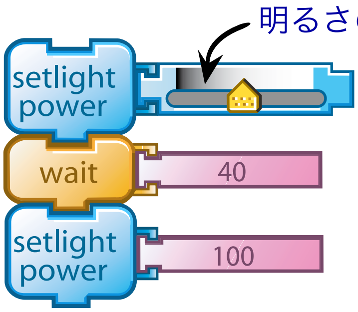
ピンク　赤　黄色　アクア　水色　紫

0　20　40　60　80　100

10　30　50　70　90

マゼンタ　オレンジ　緑　シアン　青

デフォルト値:setlightcolor 0



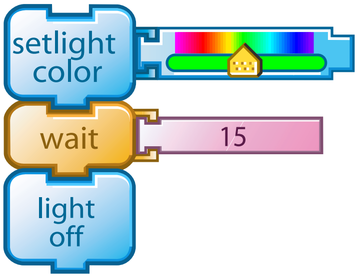
ライトを半分の明るさに設定してONにする

40ticks 待つ

ライトを最も明るい状態に戻す

setlightpower(セットライトパワー)では、0(最も暗い)～100(最も明るい)の数値を設定できます。設定する時は、スライダーか数値ブロックを使用します。

デフォルト値：setlightpower 100

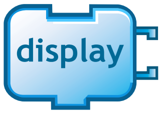


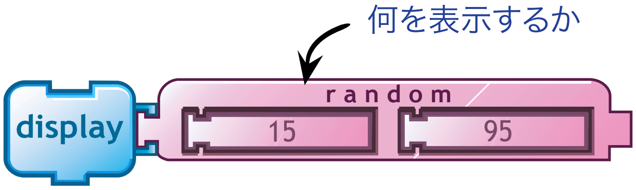
ライトを緑にする

15ticks 待つ

ライトを消す

lightoff(ライトオフ)でライトを消します。





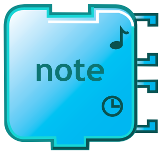
乱数をディスプレイに表示する

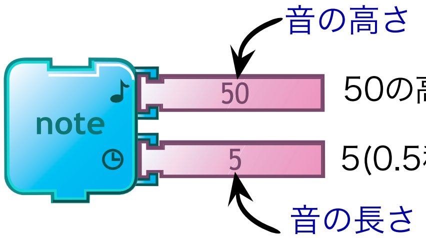
サウンドブロック

[サウンド]パレットは、6つの基本ブロック(以降に記載)から構成されています。
追加ブロックは[メロディエディタ]と[リズムエディタ]を使って作成します。
メロディとリズムの作成については、30ページを参照してください。

「チッ」と鳴らす

コンピュータがピコクリケットと通信しているかどうか確認するために、chirp(チープ)は便利です。



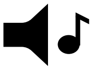


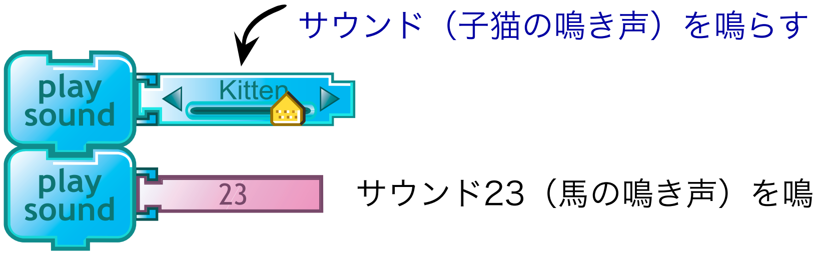
50の高さの音を鳴らす

5(0.5秒)の間

note (ノート)は0(最も低い)〜100(最も高い)の音をピコクリケットのスピーカーまたはサウンドボックスを使って鳴らします。(サウンドボックスを使用したほうが、良い音が出ます)
メロディを作曲する場合は、noteブロックを組み合わせていくよりも、[メロディエディタ]を使用することをお勧めします。

デフォルト値：**note 50 2**





サウンド23（馬の鳴き声）を鳴らす

スライダーを使用してサウンドを選択するか、0〜100の数値を入力します。
24種類のサウンドを使用できます。(次ページの表を参照)

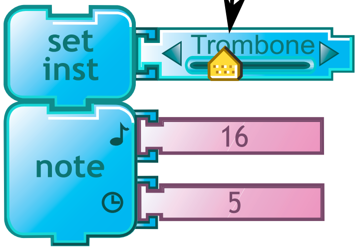
#	サウンド	#	サウンド	#	サウンド
1	ピアノ	9	拍手	17	コオロギ
2	フルート	10	カウベル	18	フェアリーダスト
3	ピチカート	11	マラカス	19	ハイQ
4	弦楽器	12	平手打ち	20	犬
5	トロンボーン	13	ティンバル	21	雄鶏
6	ヴィブラホン	14	ウッドブロック	22	ガチョウ
7	ア・ゴーゴー	15	ギロ(打楽器)	23	馬
8	カバサ	16	雑音	24	子猫

これらサウンドの再生には、サウンドボックスが必要です。サウンド0はピコクリケットの「チッ」という音です。playsound(プレイサウンド)では24より大きい数値の場合、その数値から24(またはその倍数)を引いた値の番号のサウンドを再生します。
たとえば、25はサウンド1、26はサウンド2を再生します。

デフォルト値：playsound 24

setinstrument
(セットインストルメント)

楽器を選ぶ



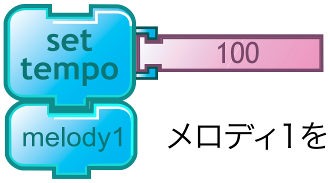
楽器をトロンボーンに設定

note 16を(トロンボーンで)鳴らす

5 ticks(0.5秒)の間

後続の単音またはメロディに使用する楽器を設定します。スライダーで楽器を選択、または0〜100の数値を指定します。setinstrument 0は単音またはメロディをピコクリケットのスピーカーに設定します。
setinstrumentの入力はplaysound(上記の表)の場合と同じです。どのサウンドも楽器として使用できますが、適しているものとそうでないものがあります。

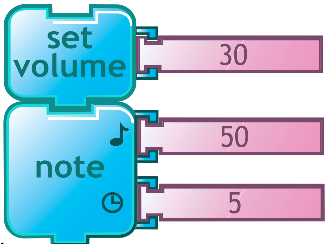
デフォルト値：setinstrument 1 (ピアノ)



テンポを100(最高速)に設定

メロディ1を最も速いテンポで鳴らす

settempoは、0(最も遅い)〜100(最も速い)の値を設定します。最も速いテンポ(100)は240 bpm（beats per minute）相当で、最も遅いテンポ（0）は60 bpm相当です。
デフォルト値：settempo 50 (120 bpm)



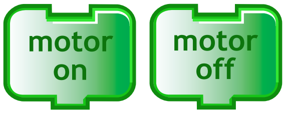
音量を30(静か)に設定

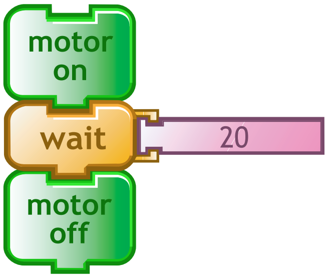
50の高さの音を

5ticks (0.5秒)の長さで鳴らす

setvolume(セットボリューム)では、0(サウンドなし)〜100(最も大きい)の値を設定します。
サウンドボックスを使用している場合のみsetvolumeは機能します。
デフォルト値：setvolume 80

アクションブロック





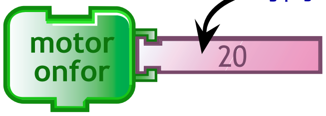
モーターをONにする

20 ticks(2秒)待つ

モーターをOFFにする

motoroff(モーターオフ)はモーターを惰性で停止させます。モーターを急に、または緩やかに停止する場合は、<u>ピコブロックステキスト言語</u>(35ページ〜)のbrake(ブレーキ)およびcoast(コースト)コマンドを使用します。

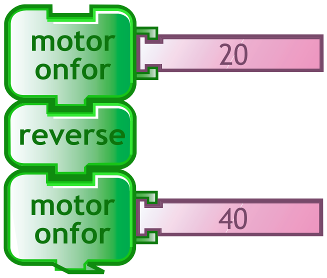
時間の長さ



モーターを20ticks (2秒)だけONにする

10 ticks = 1秒
指定したticksの間、モーターをONにし、その後OFFにします。motoronfor(モーターオンフォー)は、motoron(モーターオン)、wait(ウェイト)、motoroff(モーターオフ)というスタックと同じことを実行します。

デフォルト値： motoronfor 10

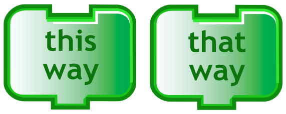


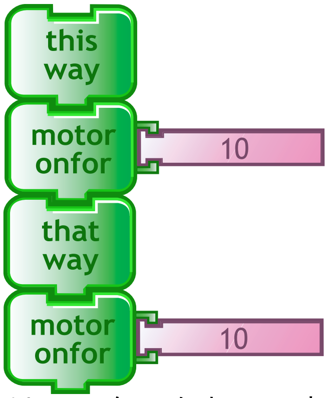
モーターを20 ticks、ONにする

回転方向を変える

モーターを（先ほどと反対方向に）
40 ticks、ONにする

reverse(リバース)は、モーターが回転する方向を変えます。
reverseはモーターの回転方向を変えるだけで、モーターをONにするものではありません。





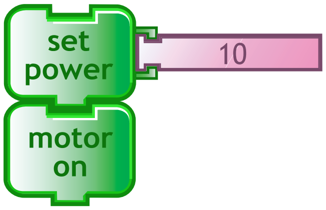
モーターの回転方向を設定

モーターを10 ticks、ONにする

モーターを反対の回転方向に設定

モーターを10 ticks、ONにする

thisway(ディスウェイ)はモーターをある一方の回転方向に設定し、thatway(ザットウェイ)はそれとは反対の回転方向に設定します。どちらがどの回転方向になるかは、モーターをどの向きで接続するかによって決まります。thiswayとthatwayはモーターをONにするものではなく、回転方向を設定するものです。



モーターの強さを10に設定する

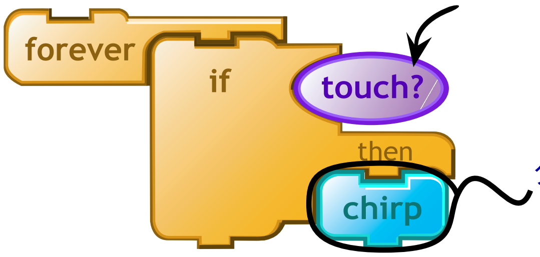
モーターをONにする

setpower(セットパワー)には、0(出力なし)～100(最大出力)の値を設定できます。setpowerはモーターをONにするものではなく、モーターの強さ（出力パワー）を設定するものです。

デフォルト値：setpower 100

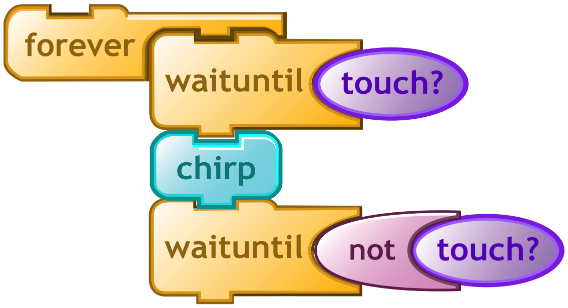
センサーブロック

タッチセンサーが押されたか



タッチセンサーが押されると、
「チッ」と鳴る

touch?(タッチ？) は、タッチセンサーが押されると、次に続くプログラムを実行します。上記の例では、タッチセンサーのボタンが押されている間、ピコクリケットが「チッ」と鳴り続けます。ボタンを押すたびに1回だけ「チッ」と鳴らす場合は、次ページのようなスタックを作成してください。

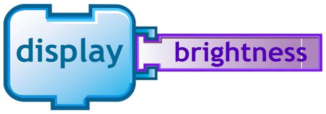


タッチセンサーが
押されるまで待つ

ライトセンサーが暗さを感知するまで待つ

この音を鳴らす(犬の鳴き声)

dark? ライトセンサーが15より小さい明るさを感知すると条件が満たされます。



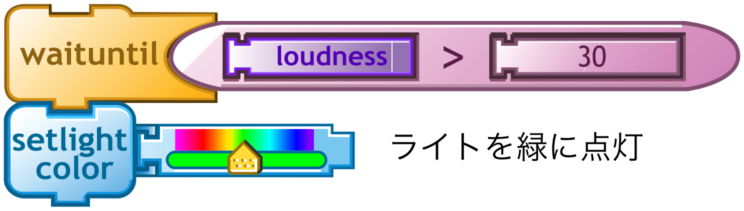
ライトセンサーが感知した数値を表示

brightness(ブライトネス)は0(光なし)から100(とても明るい)までの数値を出力します。

サウンドセンサーが音を感知するまで
待つ

ライトをピンクに点灯

サウンドセンサーが15より大きい音を感知するとloud?(ラウド?)の条件が満たされます。

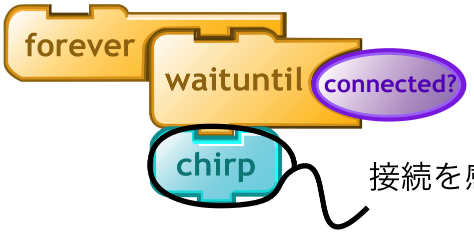


サウンドセンサーが
30より大きい音を
感知するまで待つ

ライトを緑に点灯

loudness(ラウドネス)は0(音なし)から100(大きい音)までの数値を出力します。

Ω

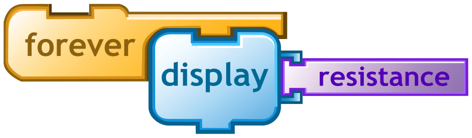


レジスタンスセンサーが
接続を検出するまで待つ

接続を感知すると、「チッ」と鳴る

レジスタンスセンサーが50より小さい抵抗を感知するとconneced? (コネクテッド?)の条件が満たされます。

Ω



レジスタンスセンサーが感知した
値を表示

resistance(レジスタンス)は0（抵抗なし）〜100（大きな抵抗）の数値を出力します。
とても狭い数値の範囲にしかならない場合、レジスタンスブロックの範囲と感度を調整すると便利です。シフトキーを押しながら、ブロックをクリックするとスライダーが表示されますので、下記のようにスライダーを調整してください。

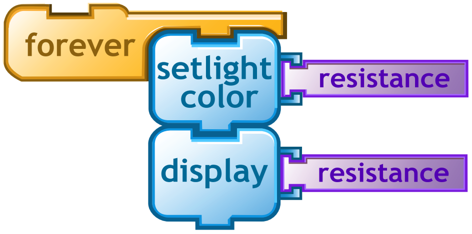
小さい数値にならない場合はこのように試してください

大きい数値にならない場合はこのように試してください

スライダーを動かしたら、[魔法のつえ]を使用してください。

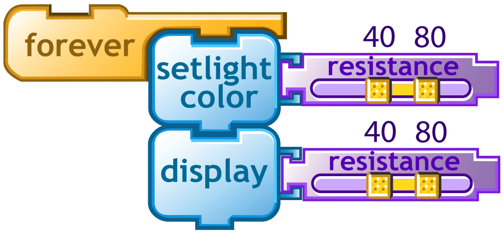
レジスタンスセンサーの値を使って、光の色をコントロールするような場合、スライダーは便利です。

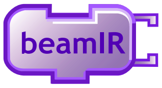
抵抗が少ししか変化しない場合、色の変化も少ししか起こりません。スライダーを使って、抵抗の変化にあわせて色の変化の幅を広げることができます。
その動作を確かめるため、何かサンプルを作ってみましょう。レジスタンスセンサーとライト、ディスプレイをピコクリケットにつなぎます。そしてレジスタンスセンサーのワニ口クリップを粘土の固まりにつなげ、プログラムをスタートさせてみましょう。

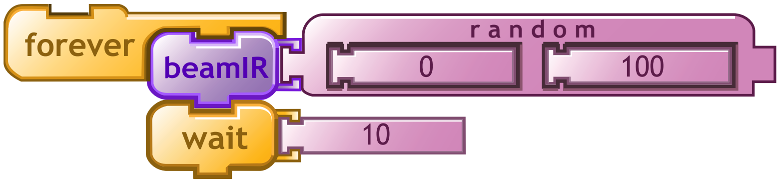


粘土をこすると、ディスプレイ上の数字が40から80辺りを変化します。
光は黄色から青／緑に変わってゆきます。

下記のように左の40の位置から右の80の間でスライダーを動かすとディスプレイの数字は
0から100まで数字の幅を広げることができます。この時、光の色は赤から紫まで変化します。

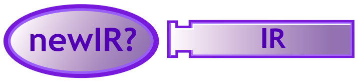


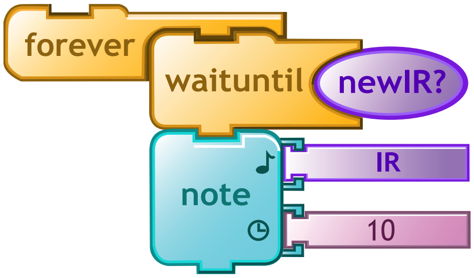




0から100までの数値を送信

beamIR(ビームIR)は赤外線（IR）で別のピコクリケットに数値を送信します。このコマンドを
使用する時は、2台のピコクリケットが必要になります。1台が赤外線信号　送信し、
もう1台が赤外線信号を受信します。IRブロックについては、下記の例を参照してください。
送信する数値は0〜100の間にしてください
範囲外の数値は予想外の結果になる場合があります。

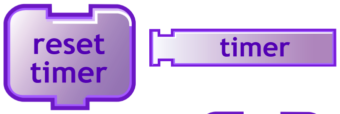


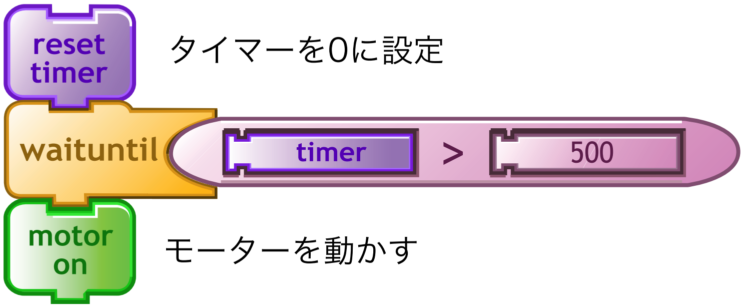


別のピコクリケットからの
新しい赤外線信号を待つ

受信した数値に基づいて音を鳴らす

IRはInfra Red（赤外線）の略語です。
新しい赤外線（IR）信号を受信すると、newIR?(ニューIR?)の条件が満たされます。
IRは最後に受信したIR信号の値を出力します。





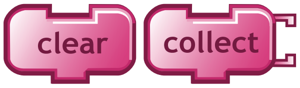
タイマーを0に設定

タイマーが500を超えるまで待つ

モーターを動かす

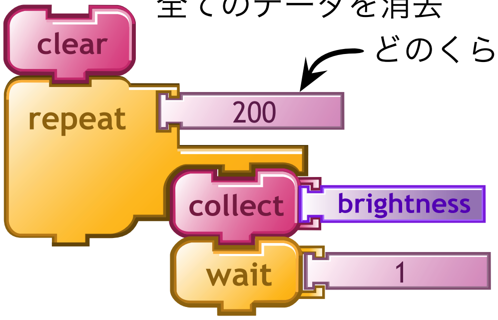
タイマーの100は1秒に相当します。

データブロック



全てのデータを消去

どのくらいデータを集めるか？

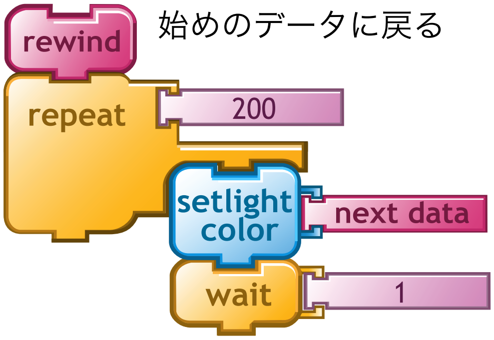


ピコクリケットのメモリーにライトセンサーから返ってくる数値を記憶する

次のデータを集めるまで0.1秒待つ

collect を使ってピコクリケットのメモリーの中にデータを貯めます。(0から100まで)
ピコクリケットは200までデータを貯めることができます。そして graph で集めたデータのグラフを見ることができます。

始めのデータに戻る



ピコクリケットのメモリーに貯められた次のデータの値をライトの色に当てはめます

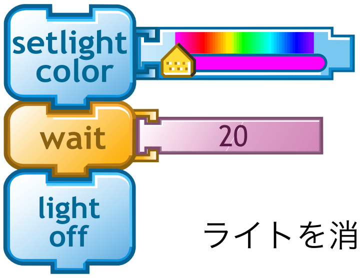
次のデータを使うまで0,1秒待つ

rewind は集めたデータの最初から使うために準備します。

nextdata はピコクリケットのメモリーにある次のデータを取り出します。データの最後まで到達したら最初の値に戻ります。

rewindブロックの下にcollectブロックをつなげると、ピコクリケットは古いデータを全て消した上で新しいデータを集めます。

フローブロック

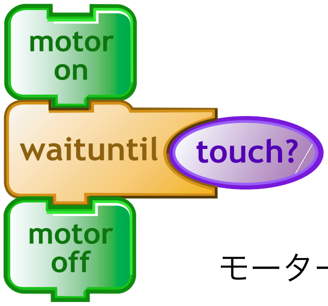


ライトをピンクに点灯

20 ticks（2秒）待つ

ライトを消す

10 ticksは1秒に相当します。
デフォルト値：**wait 10**

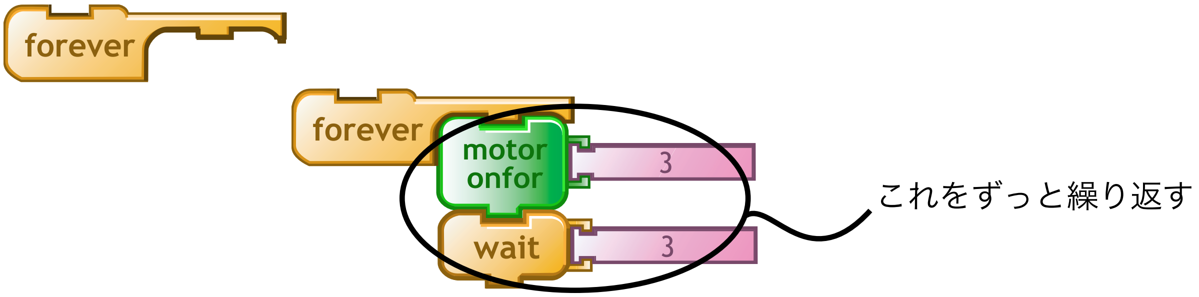


モーターをONにする

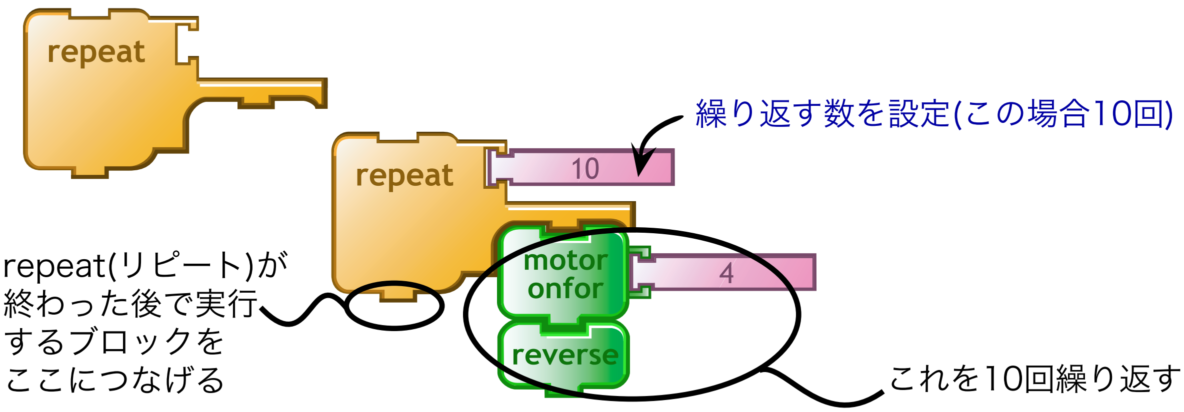
タッチセンサーが押されるまで待つ

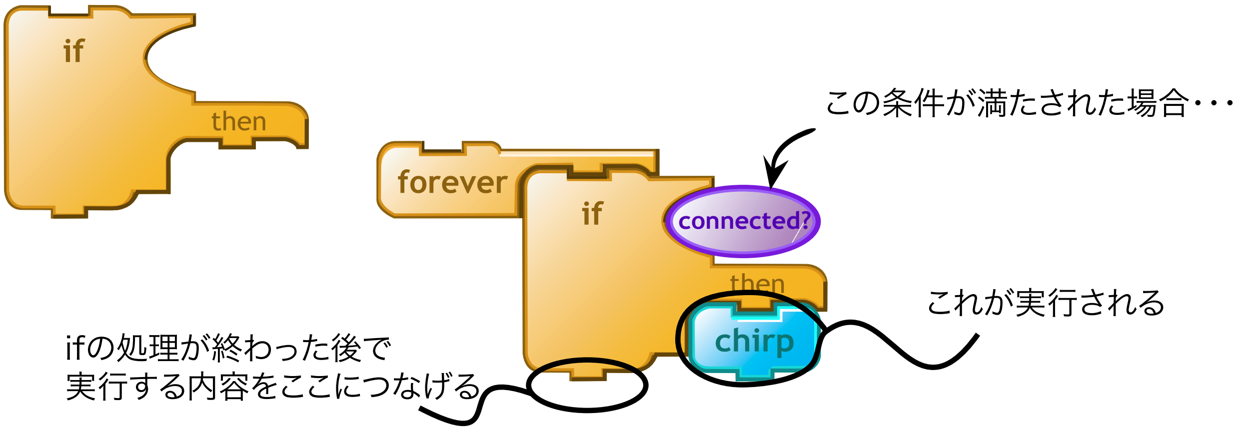
モーターをOFFにする

条件が満たされるまで待ちます。

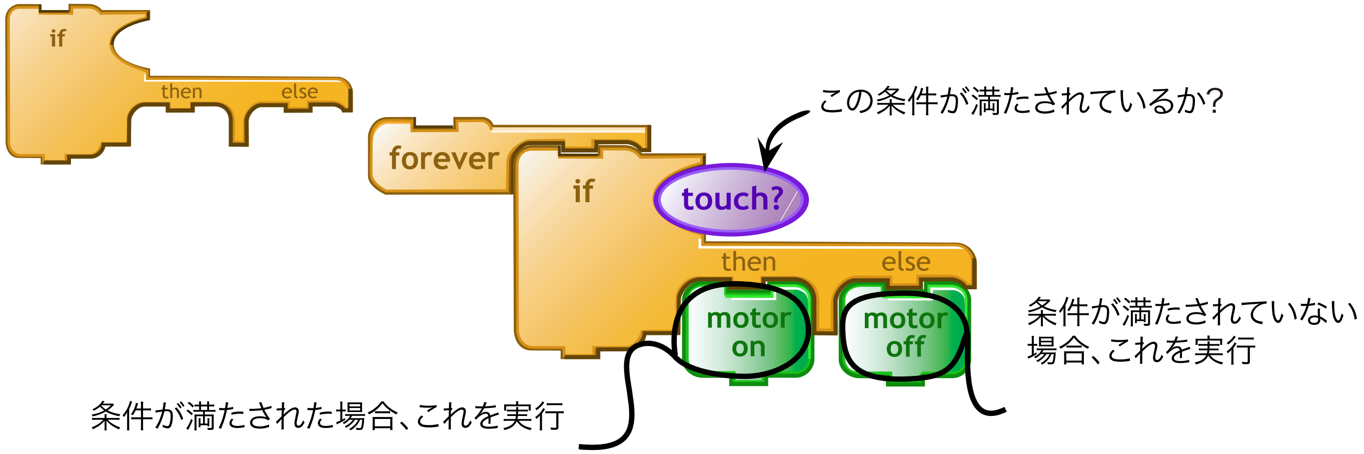


forever(フォーエバー)を停止するには、ピコクリケットのボタンを押すか、STOP をクリックします。

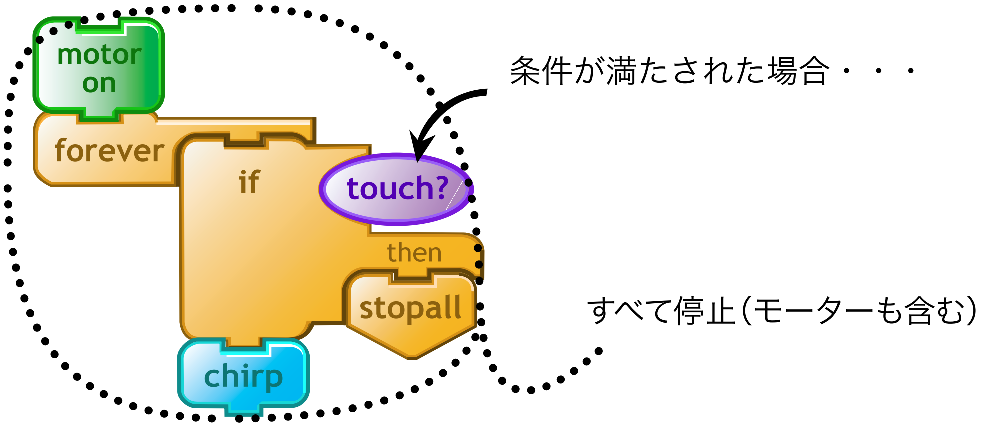




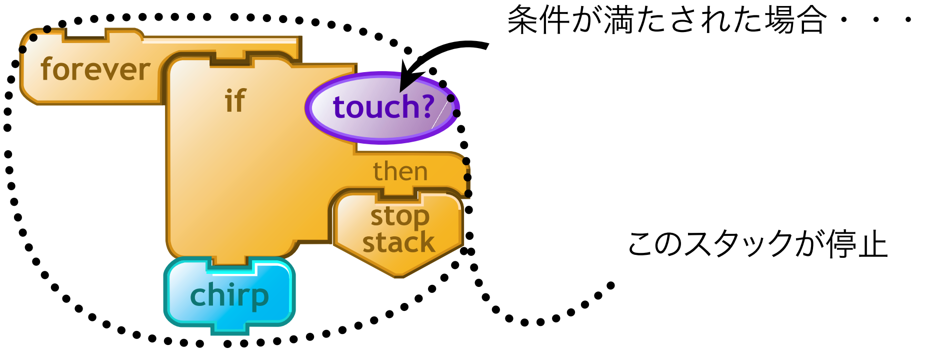
条件が満たされた場合、「then」(ゼン)に続くブロックを実行します。



条件が満たされた場合は、「then」(ゼン)という語の下にあるブロックを実行します。
条件が満たされていない場合は、「else」(エルス)の下にあるブロックを実行します。



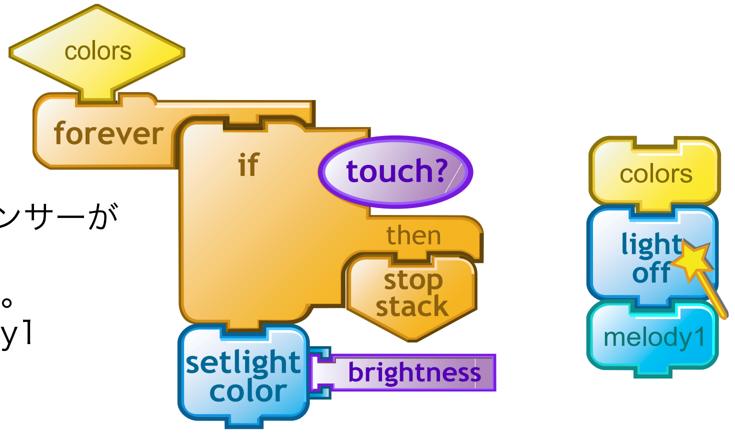
すべてのプログラムとモーターを停止します。(ライトは点灯したままになります)
stopall(ストップオール)はピコクリケットのボタンを押すことと同じです。



stopstack(ストップスタック)は、そこにつながっているスタックを停止させます。
このブロックを使用すると、他のスタックを続行しながら、あるスタックのみ停止させることができます。

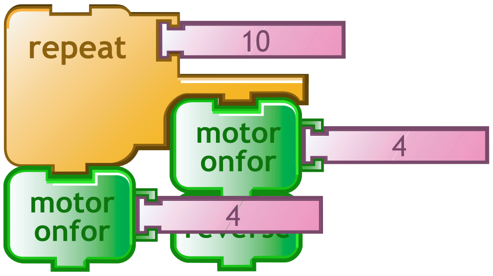
例：

colors(カラーズ)はタッチセンサーが押されると停止しますが、他のスタックは続行されます。ライトはOFFになり、melody1(メロディ1)は演奏されます。

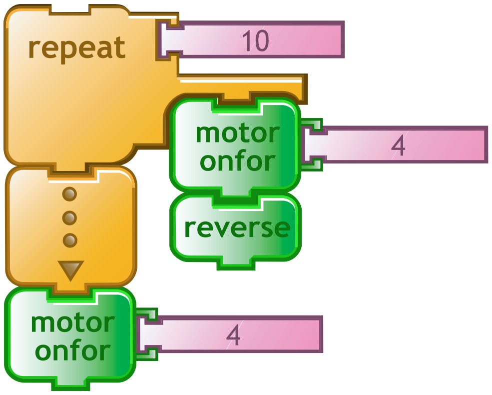


垂直スペーサー

スペーサーを使用しない場合



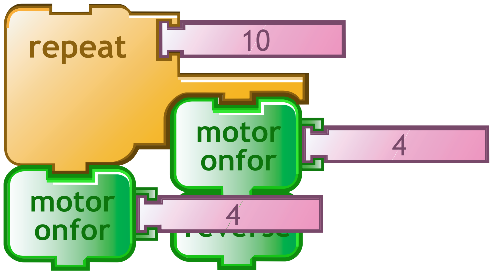
スペーサーを使用した場合



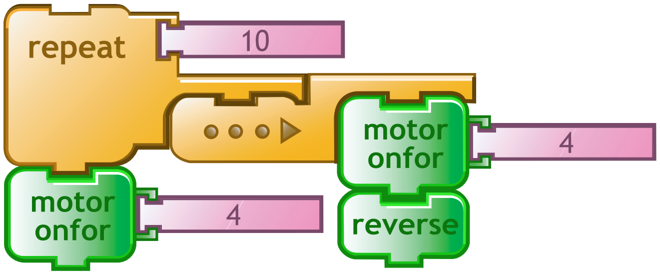
スペーサーの使用により、ブロックの重なりを避けることができます。
（スペーサーを追加しても、プログラムが実行する順番は変わりません）

水平スペーサー

スペーサーを使用しない場合

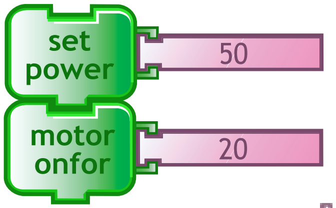


スペーサーを使用した場合



スペーサーの使用により、ブロックの重なりを避けることができます。
（スペーサーを追加しても、プログラムが実行する順番は変わりません）

ナンバーブロックス

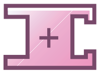


モーターの強さを50(半分の出力)に設定

モーターを20 ticks(2秒)動かす

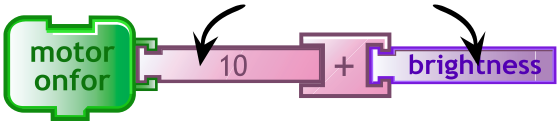
をクリックして数値を入力します。 には負の数も入力できます。

デフォルト値：10
ピコブロックスの数値は-(マイナス)32768から+(プラス)32767の整数に限られています。

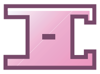


左右の値を足し算

左側に右側の値を足す

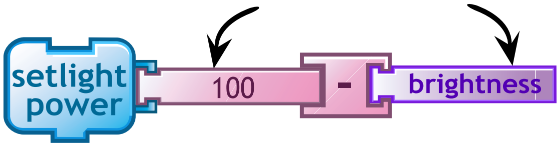


左側の「10」に右側のライトセンサーの値を足した時間だけモーターを動かします。
(たとえば、ライトセンサーの値が30の場合は、40ticksモーターをONにします)

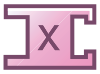


左右の値を引き算

左側から右側の値を引く

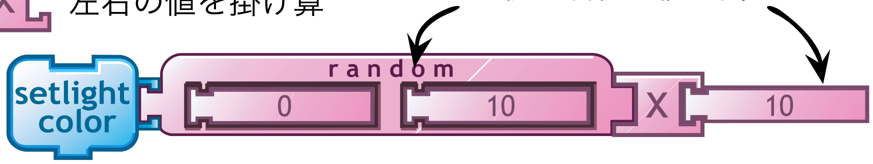


左側の「100」から右側のライトセンサーの値を引いた値でライトを点灯します。
(たとえば、ライトセンサーの値が60の場合、ライトの値は40になります)



左右の値を掛け算

左側に右側を値を掛ける



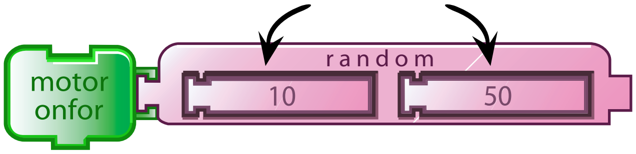
ライトの色を左側の「乱数(上記の場合0から10)」に10を掛けた色に設定します。
(たとえば、乱数が7の場合は、色を70のシアンに設定します)

左側の値を右側の値で割り算

左側を右側の値で割る

ライトセンサーの値を10で割った時間だけモーターをONにします。
(たとえば、ライトセンサーの値が50の場合は、5ticksモーターがONになります)

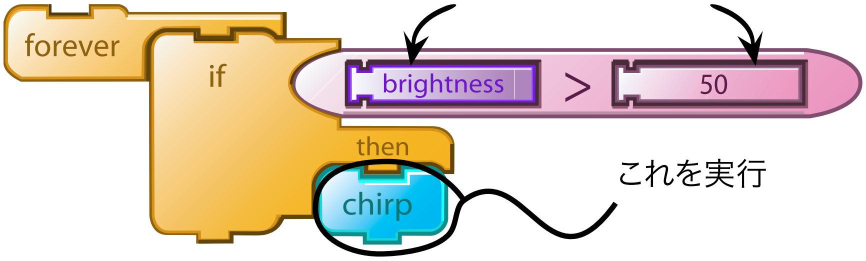
右側と左側の数の間の乱数を発生



乱数(この場合は、10か50、または
その間の任意の数)だけモーターを
ONにする

デフォルト値:random 0 100

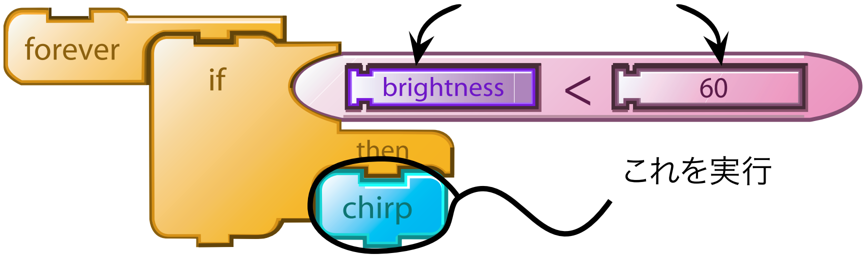
左側が右側より大きい場合...



これを実行

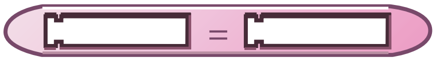
左側が右側より大きい場合、条件が満たされる

左側が右側より小さい場合...

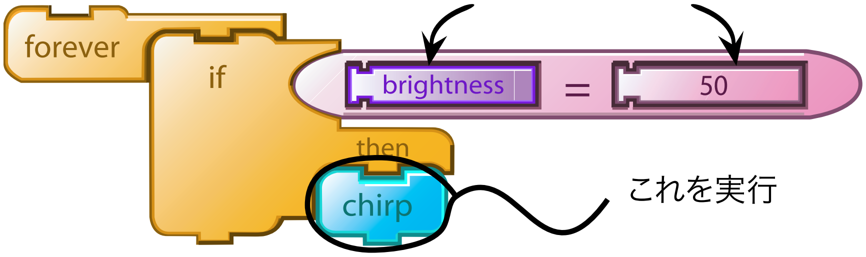


これを実行

左側が右側より小さい場合、条件が満たされる

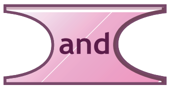


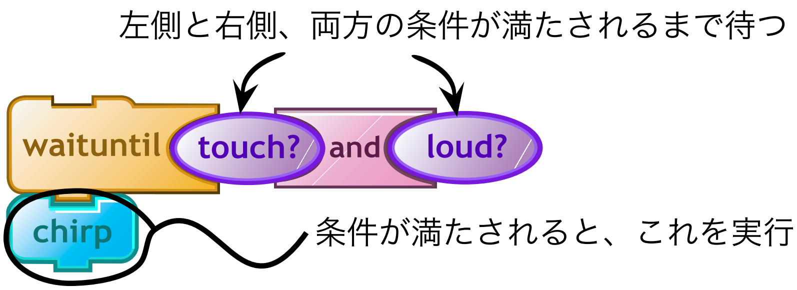
左側と右側が等しい場合...



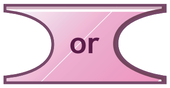
これを実行

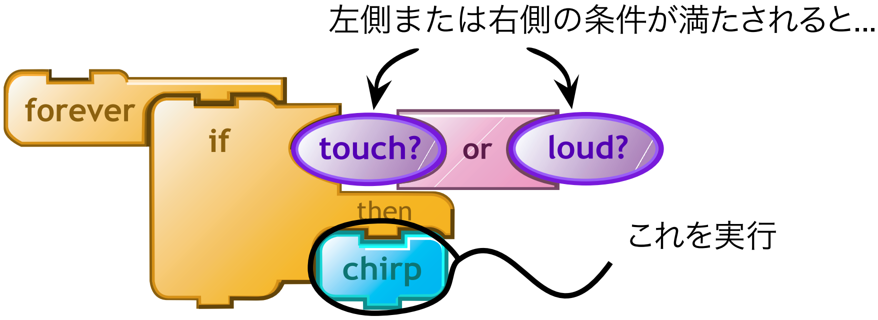
左側と右側が等しい場合、条件が満たされる



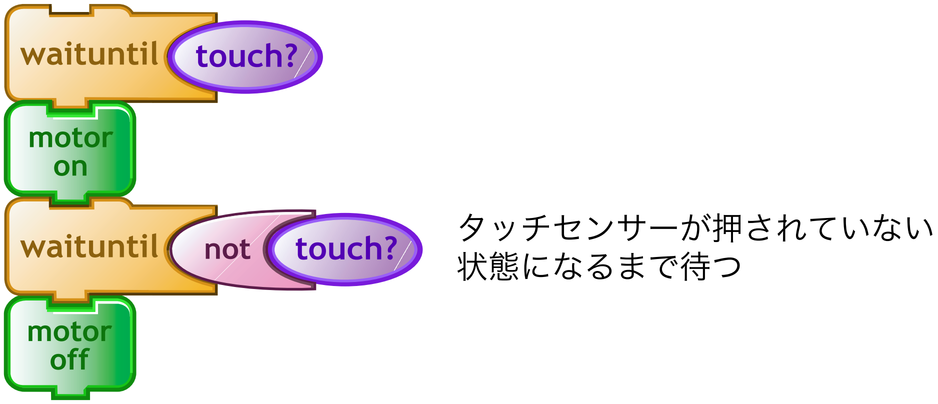


and(アンド)は、左側の条件(touch?)と右側の条件(loud?)、両方の条件が満たされると
全ての条件を満たしたと認識します。



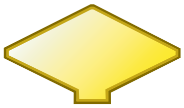


or(オア)は、左側の条件(touch?)または右側の条件(loud?)、いずれか一方の条件が
満たされると条件を満たしたと認識します。



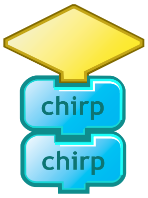
not(ノット)は、指定した条件を反対にします。(上の例を参考にしてください。)

マイブロック

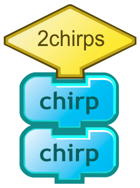


このブロックはある特定のスタックに名前を付け、そのスタックを1つのブロックとして実行させることができます。

1.名前を付けるスタックの上に、このブロックを置きます。



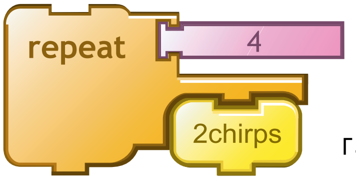
2.このブロックをクリックして、任意の名前を入力します。



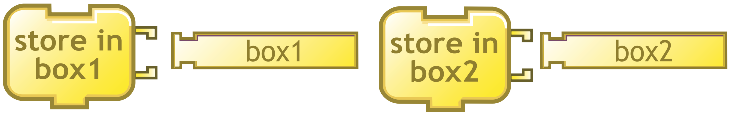
3. その名前をつけたブロックが

に表示されます。

4. このブロックは他のスタックで使用できます。



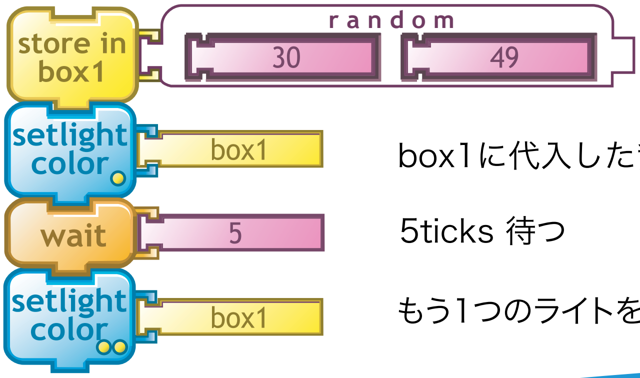
「チッ」と8回鳴る



左のブロックは、変数の値を設定したり、設定した数値を代入する際に使用します。

を使用して、値を

に代入



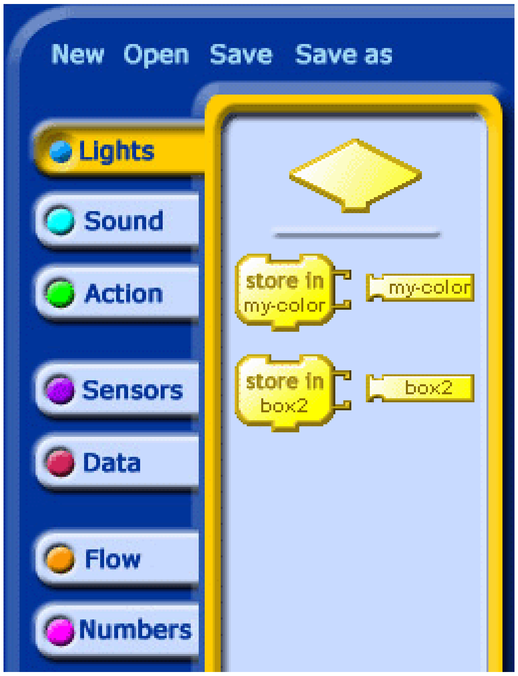
box1に乱数(左図の場合30から49までの数字)を代入

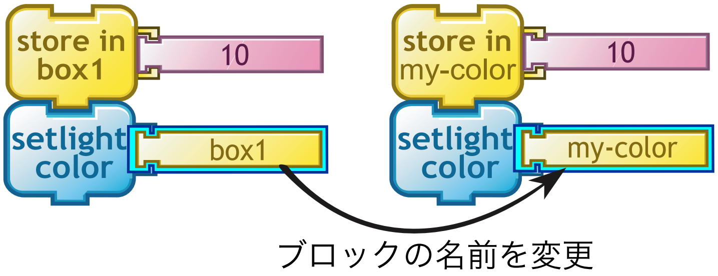
box1に代入した乱数の色でライトを点灯する

5ticks 待つ

もう1つのライトを同じ色に点灯する

変数の名前を変更する時は、その変数をワークスペースにドラッグし、クリックして新しい名前を入力します。





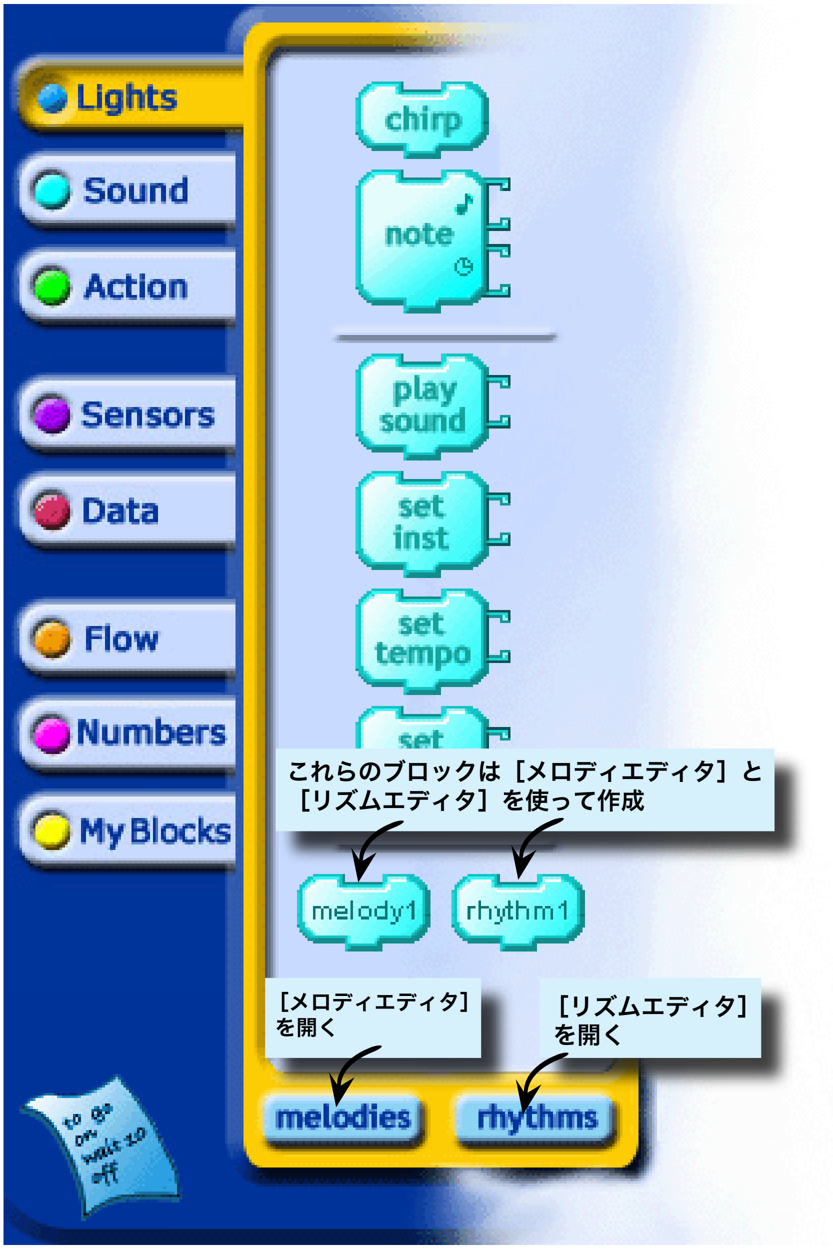
ワークスペースでbox1(ボックス1)の名前を変更すると、パレットのブロックも自動的に更新されます。

メロディエディタとリズムエディタ

ピコブロックスには、［メロディエディタ］と［リズムエディタ］という音楽用エディタが２つあります。

新しいメロディまたはリズムを作る時（あるいは既存のものを編集する場合）、まず［サウンド］タブをクリックします。［サウンド］パレットの下部には2つのボタンがあります。1つは［メロディエディタ］を開くためのボタンで、もう1つは［リズムエディタ］を開くためのボタンです。

新しいメロディまたはリズムを作成すると、［サウンド］パレットに新しいブロックとして表示されます。これらのサウンドブロックは、ピコブロックスの他のブロックと同じように、ワークスペースにドラッグして、スタックとつなげることができます。



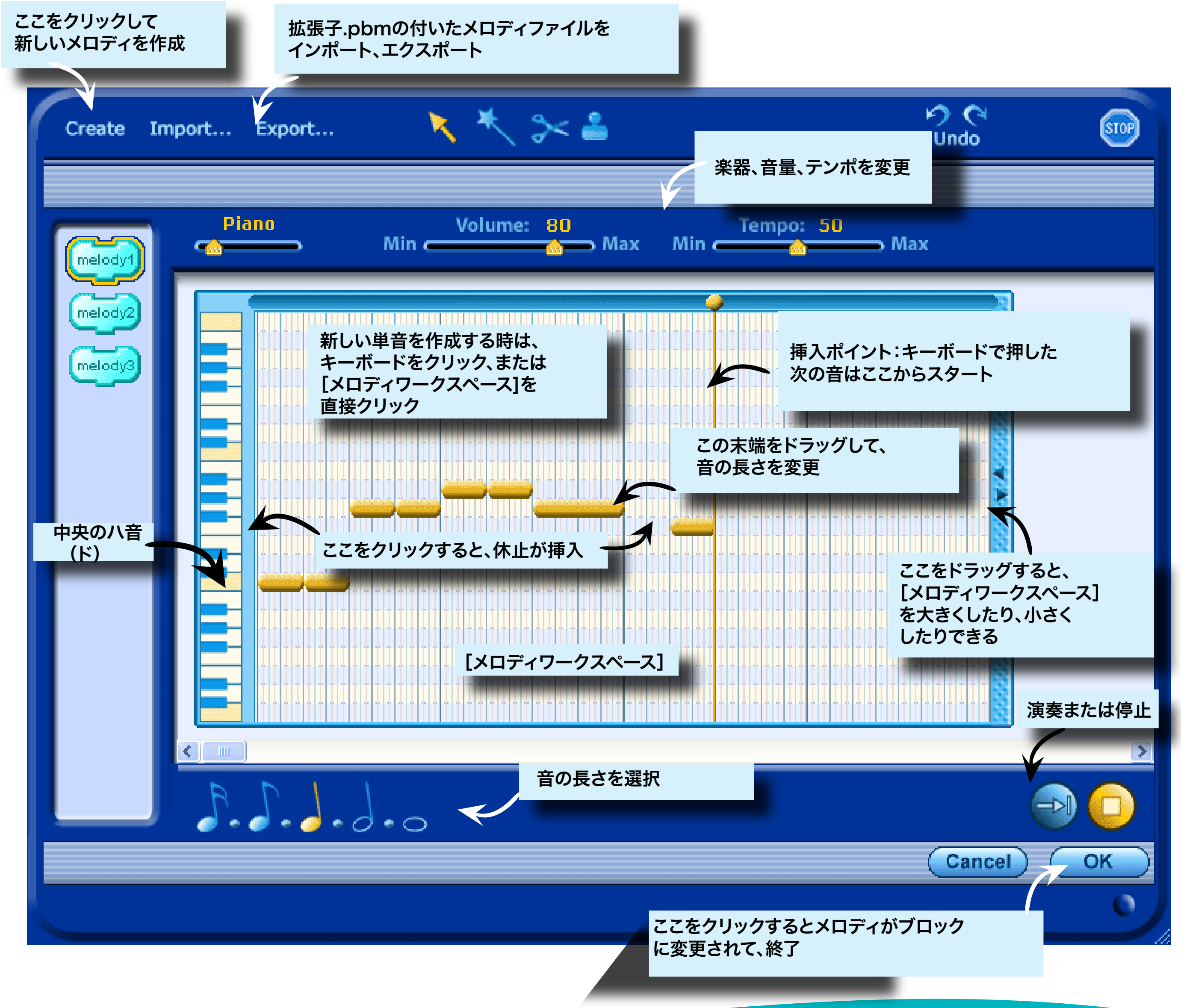
メロディエディタ

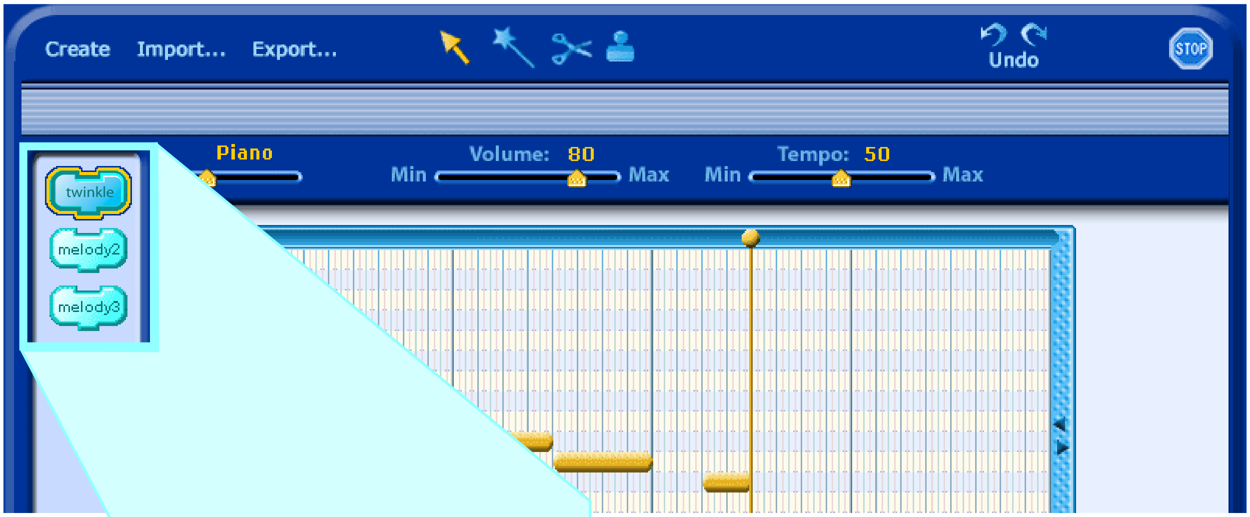
をクリックすると、[メロディエディタ]が開きます。

はじめてメロディを作る場合には、メロディを作成するための[メロディワークスペース]が何もない状態で表示されます。以前にメロディを作成していた場合には、[メロディワークスペース]に作成したメロディが表示されます。ここでは新しいメロディを作ったり（左上にある[Create(作成)]ボタンをクリックすると新しいメロディを作成）、メロディを編集したり（左側のパレットで作ったメロディを選択）することができます。メロディに音を追加する時は、鍵盤をクリックするか、[メロディワークスペース]を直接クリックしてください。

メロディ作成が終わったら、

をクリックします。
メロディがブロックに変換されて、[メロディエディタ]を終了します。

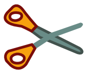


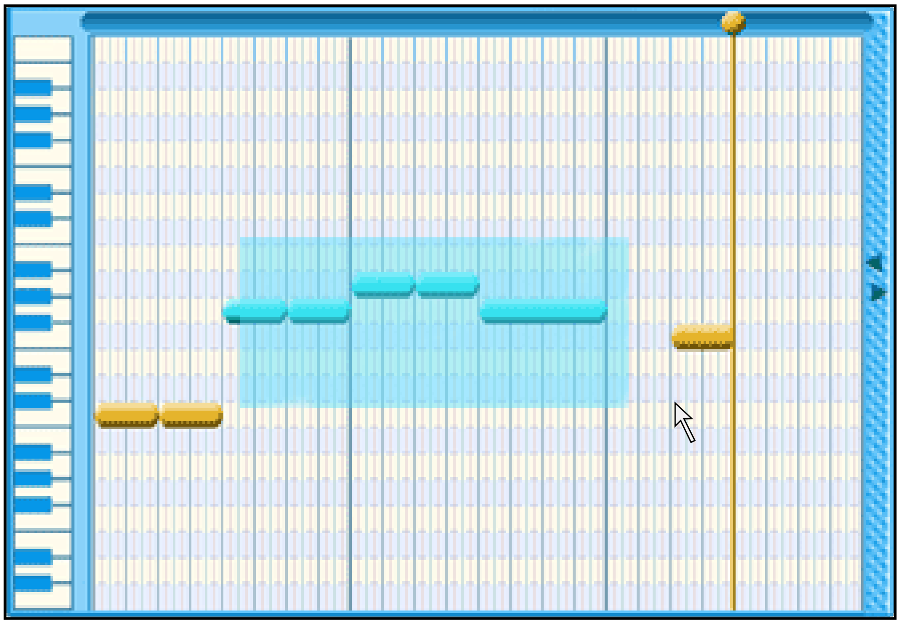


作ったメロディはすべて[メロディエディタ]の
左側に表示されます。
ここで、編集するメロディをクリックで選択します。

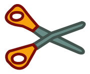
メロディの名前を変更する時は、ブロックを
クリックして、新しい名前を入力します。

を使用すると、メロディをコピーできます。

 を使用すると、メロディを削除できます。



音を選択する時は、背面をドラッグ
します。選択した音を移動する時は
他の位置にそのままドラッグします。

選択した音は、コピー 、削除
 および演奏 することが
できます。

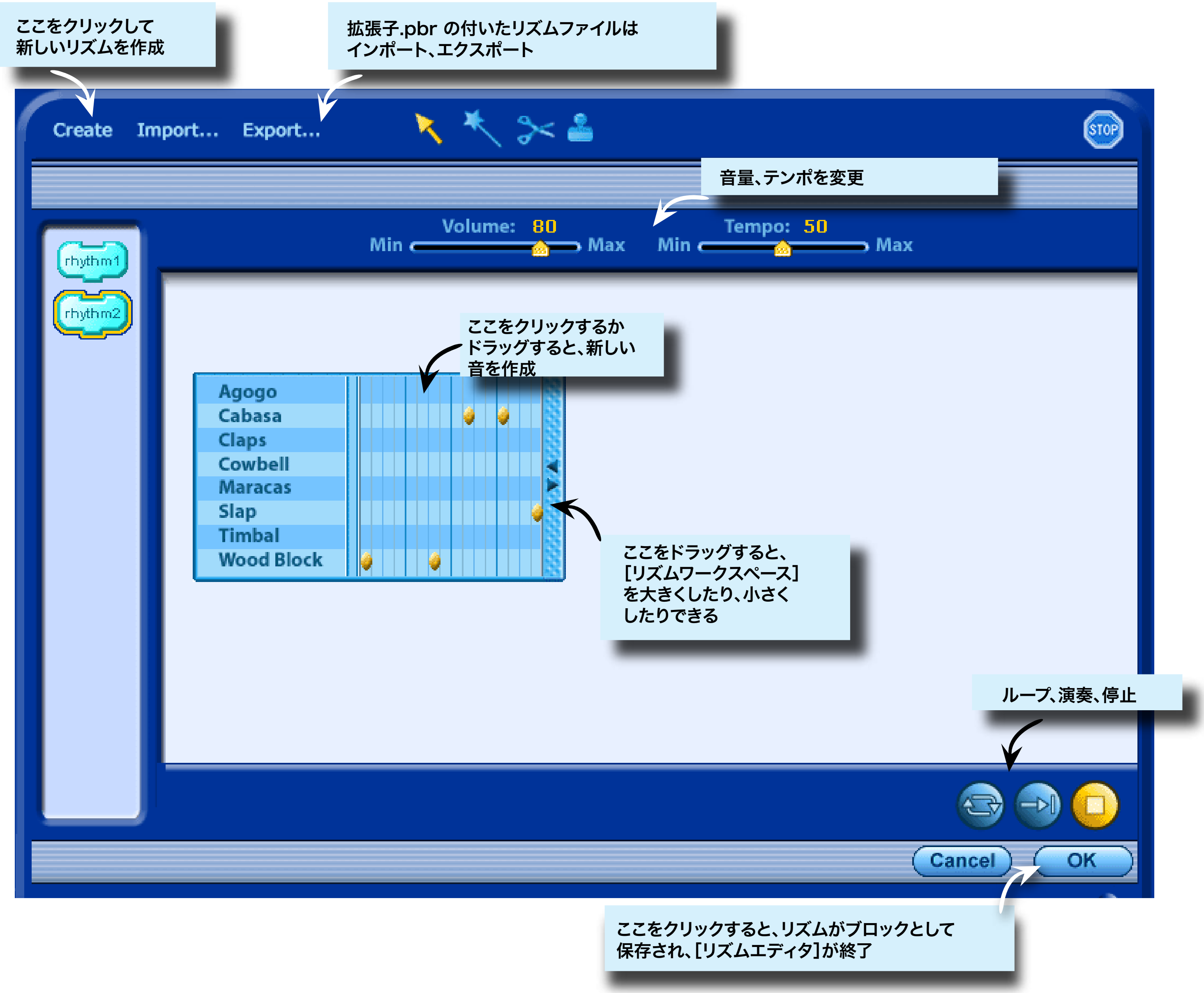
音をダブルクリックして選択すること
でも、演奏することができます。

リズムエディタ

rhythms をクリックすると、[リズムエディタ]が開きます。

はじめてリズムを作る時は、リズム作成用の[リズムワークスペース]が何もない状態で表示されます。
既にリズムを作成していた場合は、[リズムワークスペース]に作ったリズムが表示されます。
ここでは新しいリズムを作成したり(左上の[Create(作成)]ボタンで新しいリズムを作成)
リズムを編集することができます。(左側のパレット編集するリズムを選択)

リズムを作成し終えたら、 OK をクリックします。
リズムがブロックに変換されて、[リズムエディタ]を終了します。

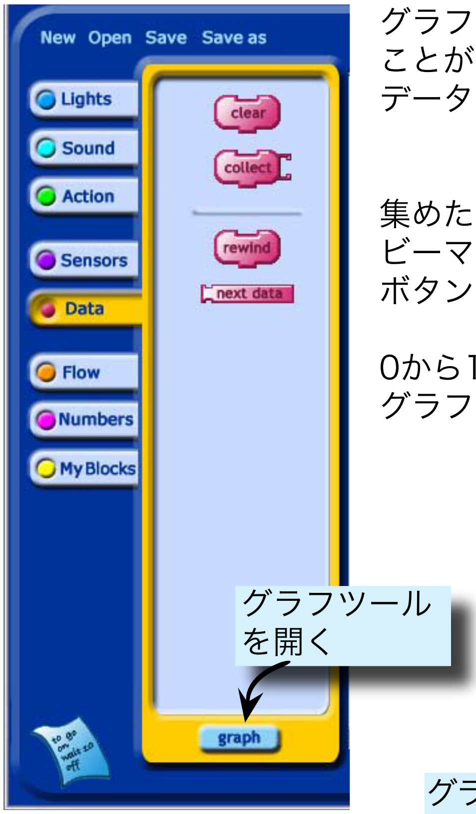


作ったリズムはすべて[リズムエディタ]の左側に表示されます。
編集するリズムをクリックで選択します。
リズムの名前を変更する時は、ブロックをクリックして、
新しい名前を入力します。

を使用すると、リズムをコピーします。

を使用すると、リズムを削除します。

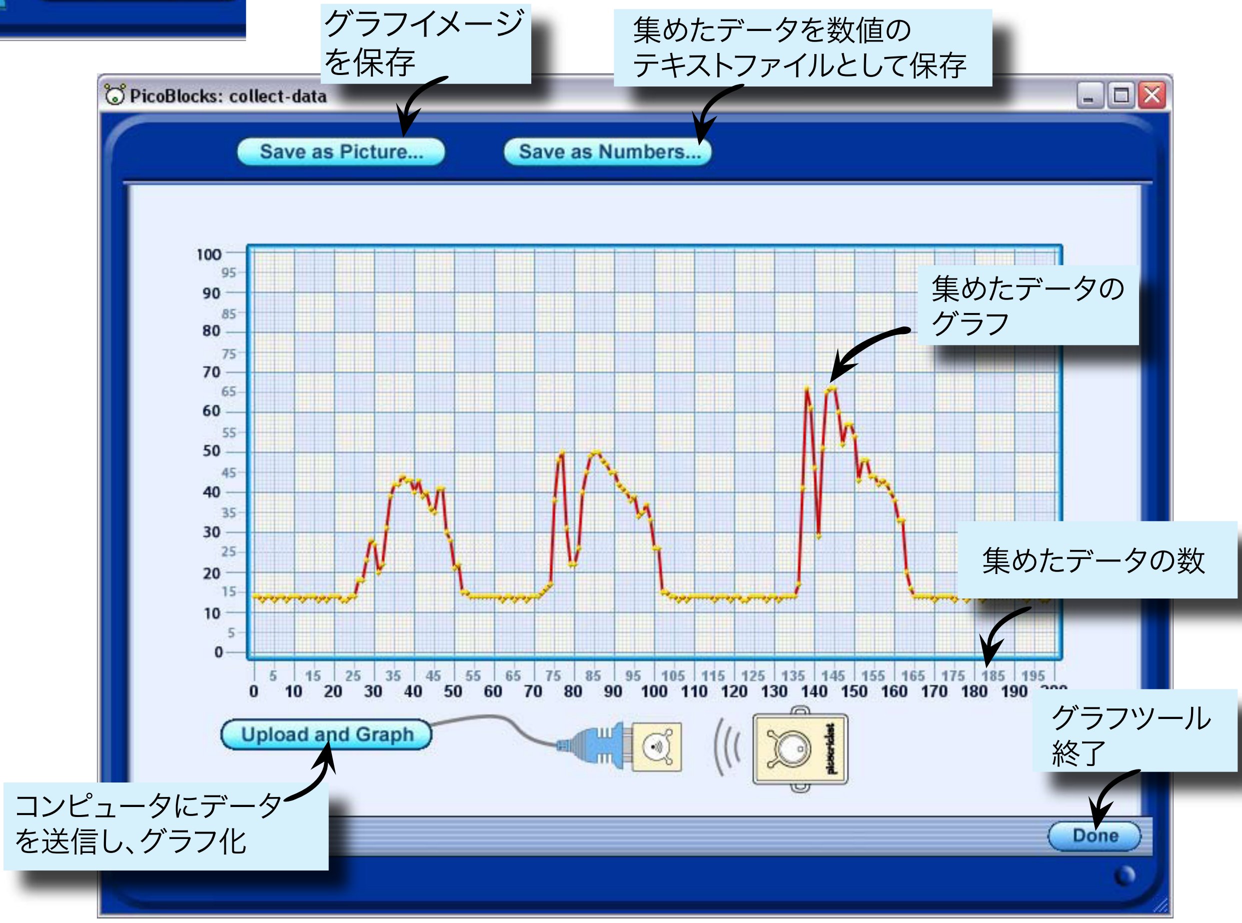
データのグラフ化



グラフツールを使ってピコクリケットから集めたデータをグラフで見ることができます。グラフツールを開くためにはデータタブをクリックし、データパレットの下にある graph ボタンをクリックします。

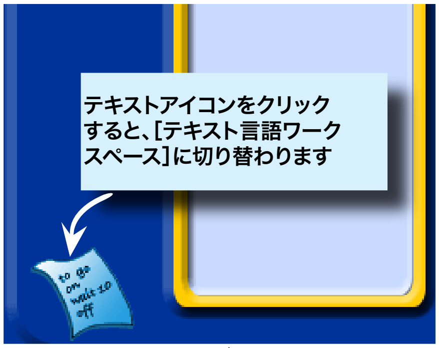
集めたデータのグラフを見るためにはピコクリケットの電源を入れ、ビーマーに向けます。そしてグラフツールの下にある Upload and Graph ボタンをクリックします。

0から100までの範囲の数値データを200個まで集め、それらのデータをグラフとして見ることができます。

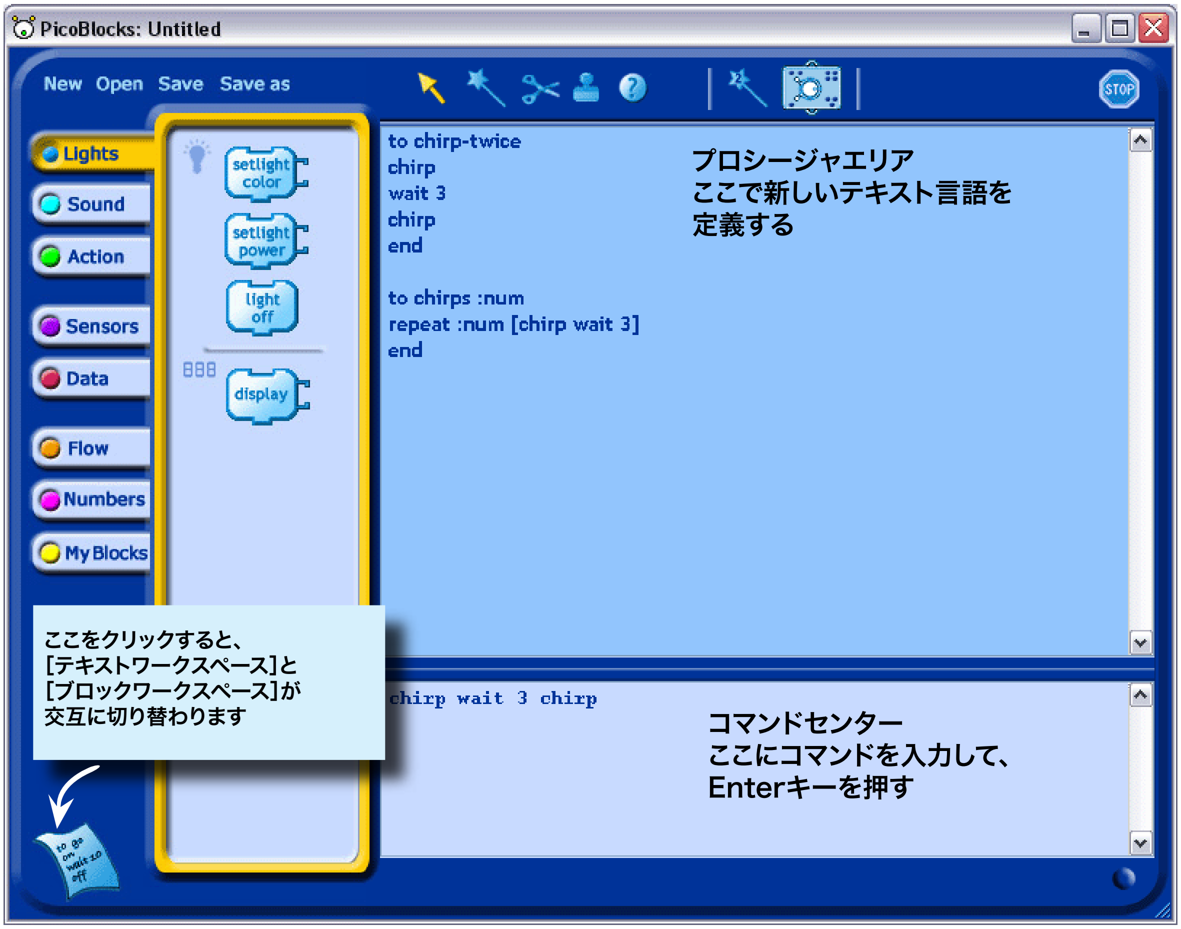


概要

ピコブロックスでは、さまざまなブロックをパズルのピースのように組み合わせることで、
コンピュータプログラムを作ることができます。ブロックには、さまざまな種類があり、
形が異なっているため、組み合わせることができるものとできないものがあります。
このため、簡単に必要なブロック同士を組み合わせることができ、ブロックの形で機能を想像する
こともできるため、プログラムを考える手助けにもなり、プログラムを作りやすくしています。
しかし、長いプログラムを作ろうすると、すべてのスタックが同時に画面に表示できなくなってしまう
などブロックが扱いにくくなることがあります。
ピコブロックスソフトウェアには、ブロックを使わない、高度な
プログラミング機能も備わっています。プログラムが長く、複雑に
なる場合は、ピコブロックステキスト言語の使用をご検討ください



ピコブロックステキスト言語はコンピュータ言語Logo（ロゴ）に
似ています。
（Logoの詳細については、www.logofoundation.org.を参照）
テキスト言語を使ってプログラムの作成をはじめる時は、
[ピコブロックスワークスペース]の左下隅にあるテキストアイコンをクリックしてください。
下図の[テキスト言語ワークスペース]が表示されます。



[コマンドセンター]に次のコマンドを入力してください。(Enterキーを押すと、各行が実行されます)

chirp	ピコクリケットが「チッ」と鳴ります!
chirp wait 2 chirp	ピコクリケットが2回「チッ」と鳴ります

テキスト言語では、スペースが非常に重要です。テキスト言語が開始・終了する場所をスペースで決定します。たとえば、上記の例で wait と 2 の間にスペースを忘れると、「I don't know how to wait2」というエラーメッセージが表示されます。これは、ピコブロックスが「wait2」というテキスト言語を認識しないためです。

ピコブロックスの各ブロックは、テキスト言語と関連しています。各ブロックとテキスト言語の関連性を見つけるためのルールはとても簡単です(1)。

ブロックに書かれている文字をそのまま使用します。ただし、スペースがある場合は削除します。

たとえば、ピコクリケットとライトをつなぎ、[コマンドセンター]に次のコマンドを入力します。

setlightcolor 20	ライトが赤に点灯する

注意:「setlightcolor」(セットライトカラー)というコマンドの中にスペースはありません。

今度は、[コマンドセンター]に次の行を入力してください。

repeat 5 [chirp wait 2]

テキスト言語では、角括弧"[ ]"を使用して、繰り返すコマンドを囲みます。特定のブロックに対応するテキスト言語の使い方を知りたい時は、知りたいブロックをクリックすると[テキスト言語ワークスペース]の左側に使い方が表示されます。たとえば、「repeat」(リピート)ブロックをクリックすると、[コマンドセンター]に次のように表示されます。

repeat 10 [WHATEVER]

この中で、「WHATEVER(ワットエバー)」は繰り返したいコマンドを入力する場所になります。

その他の例

タッチセンサーとサウンドボックスをつなぎ、次のコマンドを入力します。

waituntil [touch?] playsound 23

タッチセンサーを押すと、「ニャー」という猫の鳴き声が聞こえます。

「forever [waituntil [touch?] playsound 23]」と入力すると、今度はタッチセンサーを押すたびに「ニャー」という猫の鳴き声が聞こえます。これらの2つの例を見て、角括弧"[ ]"のある場所を確認してみてください。

ディスプレイをつなぎ、「display 7 * 9 + 6」と入力します。

テキスト言語では、丸括弧"( )"を使用して、実行する順番を決定します。次の行を入力した場合と上記の結果を比較してみてください。

display 7 * (9 + 6)

(1) テキスト言語に「then」が存在しないため、「if」および「ifelse」という文字列は例外になります。
ブロックと対応するテキスト言語のリストについては、テキスト言語のまとめ(40ページ〜)を参照してください。

プロシージャ

ピコクリケットに新しいコマンドを追加するときは、[テキスト言語ワークスペース]の
[プロシージャエリア]を使い、プロシージャを作ります。プロシージャは以下の形式で定義します。

```
to プロシージャ名
作業内容
end
```

プロシージャの定義は、はじめに「to」という文字列、続いて任意のプロシージャ名を入力します。
次に来るのはプロシージャの作業内容(プロシージャで実行するコマンドのリスト)です。
「end(エンド)」という文字列は、プロシージャの定義を完了するために使用します。
たとえば、[プロシージャエリア]に次の行を入力します。

```
to chirp-twice
chirp wait 5 chirp
end
```

これで、ピコクリケットに新しいコマンドを追加しました。[コマンドセンター]に「chirp-twice」という コマンドを
入力すると、ピコクリケットが「チッ」と2回鳴ります。

入出力のあるプロシージャ

入力または出力を行うプロシージャを作成することもできます。たとえば、「chirps」という名前で、
入力された数の回数だけchirpを繰り返す、というプロシージャを作成してみます。
入力プロシージャの場合、名前は必ずコロン「:」から始めてください。

```
to chirps :num
repeat :num [chirp wait 5]
end
```

[コマンドセンター]に「chirps 10」と入力すると、ピコクリケットが「チッ」と10回鳴ります。

「チッ」という音の回数と音の間の休止の長さを決定する2つの入力を行うプロシージャを作成することもできます。

```
to chirps2 :num :pause
repeat :num [chirp wait :pause]
end
```

[コマンドセンター]に「chirps2 10 3」と入力すると、音の間に0.3秒休止して、「チッ」と10回鳴ります。

出力プロシージャを使用すると、値を返すことができます。たとえば、「half(ハーフ)」というプロシージャを
以下のように定義します。

```
to half
output brightness / 2
end
```

ディスプレイとライトセンサーをつなぎ、[コマンドセンター]に次のコマンドを入力してください。

```
display half
```

また、「 dark? (ダーク)」というような、条件を判別するプロシージャを定義することもできます。

```
to dark?
output (brightness < 20)
end
```

ここで、[コマンドセンター]に次のコマンドを入力します。

```
forever [if dark? [chirp]]
```

十分に暗くなると、クリケットが「チッ」と鳴ります。

オリジナルブロックの作成

テキスト言語を使用すると、さまざまなブロックを作成することができ、[ピコブロックスワークスペース]で、オリジナルブロックを使用してプログラムを作ることもできます。この機能を使うことで、高度な機能が必要な場合はテキスト言語でオリジナルブロックを作成し、プログラミングは従来通りブロック言語で行うことができます。

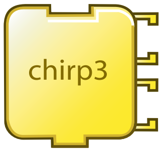
新しいブロックを定義する時は、[プロシージャエリア]に入力します。
ブロックの定義は、「block」という文字列から始めます。たとえば、次のように定義します。

block chirp1 chirp wait 3 chirp end	 2回「チッ」と鳴るブロック「chirp1」を作成

作成したブロックは自動的に”マイブロック”エリアに保存されます。オリジナルブロックには、3つまでの入力があるブロックを作ることができます。たとえば、入力が1つのブロックは下記のように定義します。

block chirp2 :num repeat :num [chirp wait 3] end	 何回も「チッ」と鳴るブロック「chirp2」を作成

また、2つの入力があるブロックは下記のように定義します。

block chirp3 :num :pause repeat :num [chirp wait :pause] end	 何回も「チッ」と鳴り、音間の休止の長さを変更できるブロック「chirp3」を作成

数値を出力するブロックを定義すると、数値ブロックの形をしたオリジナルブロックを作ることができます。

block bright2 output 2 * brightness end	 ライトセンサーが測定した明るさの2倍の数値を出力するブロック「bright2」を作成

ある条件に対する判定結果(true/false)を出力するブロックを作成する場合は、「bblock」という文字列を使用します。

bblock quiet? output loudness < 5 end	

テキスト言語におけるタグの使用

ピコブロックスの「タグ」は、同じ種類の複数の機器(2つのライトまたは2つのモーターなど)に異なる動作をさせる場合に使用します(10〜11ページのタグに関するセクションを参照)。
テキスト言語で書かれたプログラムでは、タグのないコマンドはすべての同じ種類の機器に対応します。
たとえば、2つのライトをクリケットに接続している場合、

setlightcolor 90

というコマンドで両方のライトが青く点灯します。

ポートに接続したライトのみ点灯させる場合は、コマンドの末尾にスペースなしでピリオド「.」を追加します。

setlightcolor. 90

ポートに接続したライトのみ点灯させる場合は、コマンドの末尾にスペースなしで2つピリオド「..」を追加します。

setlightcolor.. 90

ポートに接続したライトのみ点灯させる場合は、コマンドの末尾にスペースなしでピリオドとコロン「.:」を追加します。

setlightcolor.: 90

ポートに接続したライトのみ点灯させる場合は、コマンドの末尾にスペースなしで2つコロン「::」を追加します。

setlightcolor:: 90

他のコマンドにタグを付ける場合も同様に行います(2)。 たとえば、「display.: 57」は、 ポートに接続したディスプレイに数値「57」を表示します。また、「forever [if touch?: [chirp]]」は、 ポートに接続されたタッチセンサーを押すと、クリケットが「チッ」と鳴ります。

グローバル変数

「マイブロック」タブには、box1およびbox2という数値を代入するためのブロック「グローバル変数」ブロックがあります。
テキスト言語では、「global:」というキー文字列を使って最大5つのグローバル変数を追加することができます。
たとえば、[プロシージャエリア]に

global: times, pause

という行を入力すると、「times」および「pause」という2つの新しいグローバル変数が作成されます。

(2)注意：タグはサウンドボックスに関連するコマンドには使えません。

[コマンドセンター]に次のコマンドを入力してください。

storeintimes 5 storeinpause 3
repeat times [chirp wait pause]

ピコクリケットが音の間で0.3秒休止して、「チッ」という音を5回鳴らします。
またこの時、[マイブロック]タブには、新しい変数[times(タイムズ)]および[pause(ポーズ)]に対応する新しいブロックが作成されています。

ローカル変数

[make(メイク)]という文字列を使用すると、プロシージャに入力する値を変更することができます。
たとえば、[rainbow]というプロシージャを次のように定義します。

to rainbow :a
repeat 30 [setlightcolor :a make " a :a + 1 wait 2]
end

その後、ピコクリケットとライトをつなぎ、[コマンドセンター]に「rainbow 20」と入力すると、ライトが20〜49の色で点滅します。
このような変数は、作成したプロシージャに対してのみ、使用可能なため「ローカル変数」といいます。ピコブロックスではグローバル変数の数が非常に限られている(7個)ため、ローカル変数を使うと便利な場合があります。

データメモリー

ピコクリケットはデータを記憶するために200バイトのメモリがあります。[collect]と[nextdata]を使うと順番にこれらのメモリにアクセスすることができます。
(22p参照)
また、[nextdata]と[writedata]を使うと任意のアドレスにアクセスすることができます。

readata *アドレス* - メモリに記憶されている指定したアドレスの１バイトの数を読み出します

writedata *アドレス* *数値* - メモリの指定したアドレスに指定した１バイトの数を書き込みます

例えば、以下のように指定すると、
`writedata 147 75`
メモリの147番目に「75」という数字を書き込みます。これを確認するためにはピコクリケットをディスプレイにつなぎ、以下のように指定してください。

`display readdata 147`
ディスプレイに「75」という数字が表示されます。

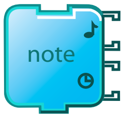
テキスト言語で使用できないツール

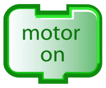
テキスト言語では「第2のつえ」が使用できません。

テキスト言語のまとめ

下記の表では、それぞれのピコブロックスに対応するテキスト言語と使用例を紹介します。

ライト		
ブロック	**テキスト言語**	**使用例**
	setlightcolor *number*	setlightcolorでは0～100までの数が使用できます。 setlightcolor 70 ライトがシアンに変わり、点灯します。
	setlightpower *number*	setlightpowerでは0～100までの数が使用できます。 setlightpower 50 ライトを半分の明るさで点灯します。
	lightoff	lightoff ライトを消します。
	display *number*	ディスプレイに数値を表示させます。 display 709 ディスプレイに 709 が表示されます。
Text only	setrgb *number number number*	ライトの色を設定します。setrgbは3色（赤、緑、青）の値を設定します。 各色の値は0～100です。 3色を合わせた色でライトが点灯します。 setrgb 0 100 100 ライトがシアンで点灯します。

サウンド		
ブロック	**テキスト言語**	**使用例**
	chirp	chirp wait 3 chirp ピコクリケットの「チッ」という音が2回鳴ります。
	note *number number*	noteには2つの入力ができます。1つ目で0～100の値で音の高さを決定し、2つ目で音の長さ（1/10秒単位）を決定します。 note 36 5 サウンドボックスで0.5秒間、音が鳴ります。
	playsound *number*	サウンドボックスを使って音を鳴らします。playsoundでは0～100の値を設定できます。サウンドボックスには、24種類のサウンドを内蔵しており、設定値が24より大きい場合、設定値から24(あるいは24の倍数)を引いた数に対応するサウンドが鳴ります。 playsound 24 サウンドボックスでニャーという猫の声が鳴ります。

サウンド		
ブロック	**テキスト言語**	**使用例**
	setinstrument *number*	noteやメロディの楽器を設定します。 setinstrumentでは0〜100の値を設定できます。 setinstrument 4 note 36 5 サウンドボックスでバイオリンの音が鳴ります。
	settempo *number*	サウンドボックスでリズムやメロディのテンポを設定します。 settempoでは0（最低速）〜100（最高速） まで設定することができます。 settempo 50 melody1 サウンドボックス標準のテンポでメロディが鳴ります。
	setvolume *number*	サウンドボックスを使って再生するnoteやサウンド、リズム、 メロディの音量を設定します。setvolumeでは0（最低音量）〜 100（最高音量）まで設定することができます。 setvolume 30 playsound 24 setvolume 100 playsound 24 サウンドボックスが最初は静かに、次に大きな音でニャーという 猫の声が鳴ります。
上記のブロックに加えて、それぞれのエディタでメロディやリズムを作成して、［サウンド］タブに独自のブロックを追加することができます。 この時作成したメロディまたはリズム名は、テキスト言語でも使用できます。		

アクション		
ブロック	**テキスト言語**	**使用例**
	motoron motoroff	モーターをON/OFFします。 motoron wait 10 motoroff モーターが1秒間だけ回転します。
	motoronfor *number*	モーターを一定時間(1/10秒単位)だけONにします。 motoronfor 10 モーターが1秒間だけ回転します。
	reverse	モーターが回転する方向を変えます。 repeat 6 [motoronfor 3 reverse] モーターが小刻みに回転/反転します。
	thisway thatway	モーターが特定の方向に回転するよう設定します。 thisway motoronfor 10 thatway motoronfor 10 モーターがある方向に1秒だけ回転し、次に反対方向に 1秒だけ回転します。
	setpower *number*	setpowerでは0〜100の値が使用できます。 setpower 100 motoronfor 10 モーターがすばやく回転します。 setpower 30 motoronfor 10 モーターがゆっくり回転します。

アクション		
ブロック	**テキスト言語**	**使用例**
テキストのみ	brake	モーターが直ちに停止します。これは、coastに続いて短いbrakeを実行するmotoroffとは動きが若干異なります。 motoron wait 10 brake モーターが1秒だけ回転し、すぐに停止します。
テキストのみ	coast	モーターの電源を切りますが、ブレーキはかけないので、モーターがゆっくり停止します。 motoron wait 10 coast モーターが1秒だけ回転し、ゆっくり停止します。

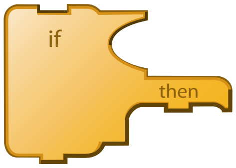
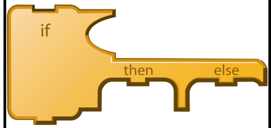
Sensors		
Block	**Word**	**Example Usage**
touch?	touch?	タッチセンサーが押された場合、条件が満たされます。 waituntil [touch?] chirp ピコクリケットは、タッチセンサーが押されるまで待ち、「チッ」と鳴ります。
dark?	dark?	ライトセンサーの値が15より小さい場合、条件が満たされます。 waituntil [dark?] chirp ピコクリケットが暗さを感知すると「チッ」と鳴ります。
brightness	brightness	ライトセンサーが測定した0～100の値を返します。 forever [display brightness wait 1] ライトセンサーが測定する明るさを継続的にディスプレイに表示します。
loud?	loud?	サウンドセンサーが測定した値が15より大きい場合条件が満たされます。 waituntil [loud?] chirp サウンドセンサーが大きな音を感知すると、ピコクリケットが「チッ」と鳴ります。
loudness	loudness	サウンドセンサーが測定した値を0～100の値で出力します。 forever [display loudness] サウンドセンサーが測定する音量を継続的にディスプレイに表示します。
connected?	connected?	レジスタンスセンサーが測定した値が50より小さい場合条件が満たされます。 waituntil [connected?] chirp ピコクリケットは、レジスタンスセンサーのワニ口クリップが接続される「チッ」と鳴ります。

センサー		
ブロック	**テキスト言語**	**使用例**
	resistance	0〜100の値を出力します。 テキスト言語では、スライダーを使用できません。 forever [display resistance wait 1] レジスタンスセンサーが測定する電気抵抗を継続的にディスプレイに表示します。
テキストのみ	rawresistance	レジスタンスセンサーが測定した値を0〜1023の数値で出力します。 rawresistanceはresistanceより細かい値を表示します。 forever [display rawresistance wait 1] レジスタンスセンサーが測定する電気抵抗を0〜1023の範囲で表示し、継続的にディスプレイに表示します。 （1000以上の数値はダッシュ記号”-”と表示されます）。
テキストのみ	rawrange	rawrangeはレジスタンスセンサーの感度を調整する時に使用します。 これには0、1または2のいずれかを入力するよう設定します。 0は最高の感度になるので、1000オーム以下の抵抗を測定する時に使用してください。1は中程度の感度になり、これがデフォルト設定です。約10,000オームの抵抗で最もよく機能します。2は感度が低くなりますので、100,000オーム以上の抵抗を測定する時に使用してください。 rawrange 0 display rawresistance
	beamir *number*	ピコクリケットの赤外線送信機から数値を発信します。 beamir 4
	newir?	ピコクリケットの赤外線受信機が新しい値を受信すると、条件が満たされます。 waituntil [newir?] chirp ピコクリケットが新しい数値を受信すると、「チッ」と鳴ります。
	ir	ピコクリケットに赤外線送信された最新の数値を出力します to test waituntil [newir?] if ir = 10 [chirp] end 送信された値が10の時、ピコクリケットが「チッ」と鳴ります。
	resettimer	ピコクリケットの内蔵タイマーの値をゼロにリセットします。 下記、timerの例を参考にしてください。
	timer	ピコクリケットのタイマー値を出力します。タイマー値は1/100秒単位で時間を測定します。 to test resettimer chirp waituntil [touch?] display timer end ピコクリケットが「チッ」と鳴ってから、タッチセンサーが押されるまでに経過した時間がディスプレイに表示されます。

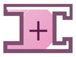
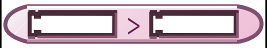
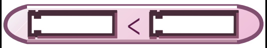
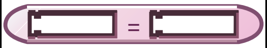
データ		
ブロック	テキスト言語	使用例
	`clear`	継続的に集めた全てのデータを消去します。新しいデータを集める前は[clear]をお使いください。以下、例をご参照ください。
	`collect` *number*	ピコクリケットにデータを集めます。(0から100の値) clear repeat 200 [collect loudness]
	`rewind`	データの一番始めの値から集めたデータをピコクリケットで再生させる準備をします。 rewind repeat 200 [display next data wait 5]
	`nextdata`	ピコクリケットに記憶された次のデータを出力します。 next data repeat 200 [display next data wait 5]
text only	`datalength`	集められた全てのデータを出力します。 display datalength 47 75
text only	`writedata` *address number*	指定の場所に１バイトのデータを書き込みます。 writedate 47 75
text only	`readdata` *address*	指定の場所の１バイトのデータを出力します。 display readdate 47

フロー		
ブロック	**テキスト言語**	**使用例**
	wait *number*	指定時間(1/10秒単位)待ちます。 chirp wait 10 chirp ピコクリケットが2回「チッ」と鳴る間に1秒だけ停止します。
テキストのみ	mwait *number*	指定時間(1/1000秒単位)待ちます。 setlightcolor 20 mwait 10 lightoff ライトが非常に早く点滅します。
テキストのみ	no-op	約40マイクロ秒間、何も実行しません。 repeat 10 [no-op] ピコクリケットが約400マイクロ秒間待ちます。
	waituntil [*condition*]	条件が満たされるまで待ちます。 waituntil [touch?] chirp ピコクリケットはタッチセンサーが押されるまで待ち、「チッ」と鳴ります。

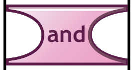
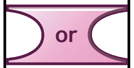
フロー

ブロック	テキスト言語	使用例
	repeat *number* [*list*]	命令を指定の回数だけ繰り返します。 repeat 5 [chirp wait 2] ピコクリケットが5回「チッ」と鳴ります。
	if *condition* [*list*]	指定の条件が満たされると、命令を実行します。 forever [if touch? [chirp wait 1]] タッチセンサーを押すと、ピコクリケットが「チッ」と鳴り、押さないと鳴りません。
	ifelse *condition* [*list1*] [*list2*]	指定の条件が満たされると最初の命令を実行し、満たされないと2番目のリストを実行します。 to test forever [ifelse touch? [setlightcolor random 0 99] [lightoff]] end タッチセンサーを押すと、ライトがさまざまな色に点滅し、押さないとライトが消えます。
	stopall	プログラムやモーターなど、すべて停止します。 stopall（ストップオール）はピコクリケットのボタンを押すことと同じです。 to test motoron forever [if touch? [stopall] setlightcolor random 0 99] end タッチセンサーを押すと、ライトとモーターが停止します。
	stopstack	プロシージャ(該当するスタック)の実行を終了します。 to test forever [if touch? [stopstack] setlightcolor random 0 99] end
		テキスト言語では、スペーサは必要ありません。
		テキスト言語では、スペーサは必要ありません。

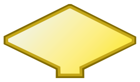
ナンバー

ブロック	テキスト言語	使用例
	number	数値を出力します。ピコブロックスの数値は -32768～+32767の整数です。 display 7　ディスプレイに数値の7が表示されます。
	*number* + *number*	2つの数値の合計を出力します。 display 7 + 4 ディスプレイに数値の11が表示されます。
	number - *number*	2つの数値の差を出力します。 display 7 - 4 ディスプレイに数値の3が表示されます。
	number * *number*	2つの数値の積を出力します。 display 7 * 4 ディスプレイに数値の28が表示されます。
	number / *number*	2つの数値の商を算出し、端数を切り捨てた値を出力 します。 display 7 / 4 ディスプレイに数値の1が表示されます。
テキストのみ	%	2つの数値を割り算した剰余を出力します。 display 7 % 4 ディスプレイに数値の3が表示されます。
	random *number* *number*	下限(左側の入力値)と上限(右側の入力値)の2つの入力から 乱数を出力します。 display random 0 10 ディスプレイに0～10の乱数が表示されます。
	*number* >	左側の入力値が右側の入力値より大きい場合、 条件が満たされます。 to test forever [if brightness > 30 [chirp]] end ライトセンサーが30より大きな値を感知すると、 ピコクリケットが「チッ」と鳴ります。
	*number* < *number*	左側の入力値が右側の入力値より小さい場合、 条件が満たされます。 to test forever [if brightness < 30 [chirp]] end ライトセンサーが30より小さな値を感知すると、 ピコクリケットが「チッ」と鳴ります。
	*number* = *number*	左側の入力値が右側の入力値と等しい場合、 条件が満たされます。 forever [if brightness = 30 [chirp]] ライトセンサーが「30」という値を感知すると、 ピコクリケットが「チッ」と鳴ります。

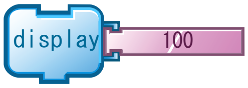
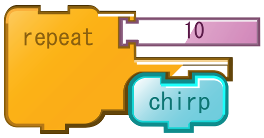
ナンバー

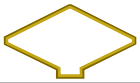
ブロック	テキスト言語	使用例
	*condition* and *condition*	左側の条件と右側の条件の両方を満たしている場合、 条件が全て満たされたとみなします。 to test forever [if (brightness < 30) and touch? [chirp]] end ライトセンサーが30より小さい値を測定および タッチセンサーが押された場合に、ピコクリケットが 「チッ」と鳴ります。
	*condition* or *condition*	左側または右側のいずれか一方の条件を満たして いる場合、条件が満たされたとみなします。 to test forever [if (brightness < 30) or touch? [chirp]] end ライトセンサーが30より小さい値を測定またはタッチ センサーが押されると、ピコクリケットが「チッ」と鳴ります。
	not *condition*	条件を反対にします。 to test forever [if not touch? [chirp]] end タッチセンサーが押されないと、ピコクリケットが「チッ」 と鳴ります。

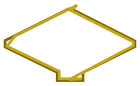
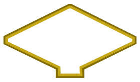
マイブロック

ブロック	テキスト言語	使用例
	`block` *`name`* *`block-definition`* `end`	テキスト言語を使って、独自のブロックを作成します。 （オリジナルブロックの作成セクションを参照）
	`storeinbox1` *`number`*	box1というグローバル変数に数値を代入します。 storeinbox1 57 display box1 ディスプレイに数値の57が表示されます。
	`box1`	box1というグローバル変数に代入された数値を出力します。 storeinbox1 57
テキストのみ	`global:`	テキスト言語を使って、新しいグローバル変数を作成します。 （前述のグローバル変数セクションを参照）
テキストのみ	`make "`*`name value`*	ローカル変数の値を変更します。 （前述のローカル変数セクションを参照）

一般的なエラーメッセージ

エラーメッセージ	説明
ビーマーがピコクリケットを見つけられません	ビーマーはプログラムを送信しているが、ピコクリケットが応答していません。 解決策：ピコクリケットの電源が入っていること、ピコクリケットとビーマーが互いに向き合っていることを確認してください。 詳細については、 www.picocricket.com/troubleshootingを参照してください。
ビーマーが見つかりません	ピコブロックスで通信ポートを開くことができません。 解決策：ビーマーがコンピュータに適切に接続されていることを確認してください。Windowsを使用している場合は、実行しているピコブロックスが1つだけであることを確認してください。 詳細については、www.picocricket.com/supportを参照してください。
空のスコアは保存できません	エクスポートしようしたメロディまたはリズムが空です。 解決策：メロディ、リズムブロックの楽譜が空になっていないか確認し、[エクスポート]を再度クリックしてください。
_____ は名前に使えません	ピコブロックスでは、その文字列を名前として使用することができません。 数値、% または [で始まる名前および既存の文字列は無効と見なされます。 解決策：別の名前を入力してください。
ダウンロードエラー	ピコクリケットにプログラムをダウンロードしているときに、通信トラブルが発生しました。 解決策：ピコクリケットとビーマーが互いに向き合っていることを確認し、もう一度試してください。ダウンロードしていないのにピコクリケットの黄色いアンテナライトが点滅している場合は、ピコクリケットがモニタや他の機器から赤外線干渉を受けている場合があります。 ピコクリケットをモニタやその他の機器から遠ざけてください。詳細には、www.picocricket.com/troubleshootingを参照してください。
______ への入力がありません	1つ（またはそれ以上）のブロックに入力が必要です。 解決策：入力を必要とするすべてのブロックに入力ブロックがついているか確認してください。 （入力が必要かどうかは、ブロックの形状で分かります） たとえば、ディスプレイには入力として数値（または他の値）のブロックをつける必要があります。  repeatでは、右上に数値が必要です。また、「アーム」の下に、繰り返したいブロックを1つ以上 つなげる必要があります。  ヒント：ブロックに入力が必要かチェックする時は[ヘルプ]ツールで ブロックをクリックしください。

エラーメッセージ	説明
_______ のやり方が分かりません	ピコブロックスが、存在しない名前を実行しようとしました。 解決策：スタックに名前を付けると、ピコブロックスがその名前のブロックを作成します。このブロックを実行するためには、[ワークスペース]上に名前を付けたスタックを置いておく必要があります。
_____ はすでに存在します	同じ名前のブロックがすでに存在します。 解決策：別の名前を入力してください。
ナンバーブロックに ______ は入れられません	に無効な文字を入力しました。 解決策：数値のみ入力してください。
プロジェクトにブロックが 多すぎます	プログラムに含まれているブロックが多すぎます。 解決策：使用しないスタック、メロディおよびリズムをすべて削除してください。
スタックにブロックが多すぎます	スタックに含まれているブロックが多すぎます。 （または、使用するメモリが多すぎます）。 解決策：repeat などを使用して、少ないブロックで多くのことを実行できるようにしてください。あるいは、スタックの上に  を配置して、マイブロックを作成してください。
ミュージックブロックが 多すぎます： 24個まで	最大数を超えた音楽ブロックを作成しました。 メロディおよびリズムの最大数は24ブロックです。 解決策：プログラムする音楽ブロックが合計24以下になるように、メロディまたはリズムブロックをいくつか削除してください。
1つのミュージックブロックに ノートが多すぎます	現在のメロディまたはリズムが長すぎます。 解決策：メロディまたはリズムを2つのブロックに分けてください。 まず、現在のメロディまたはリズムブロックをコピーします。 楽譜の前半を削除したブロックと後半を削除したブロックを作成し、これら2つのブロックを一緒に使用して、一つメロディやリズムとして再生します。
waituntil は入力が必要です	waituntilブロックには、適合する入力ブロックが必要です。 解決策：waituntilブロックの右側に適合するブロックをつけてください 例： または

エラーメッセージ	説明
空のミュージックブロックは コピーできません	コピーしようとしたメロディまたはリズムが空です。 解決策：楽譜をつくってから、ブロックをコピーしてください。
スタックに______という名前は つけられません	 に入力した名前に、無効な文字が含まれています。 %　“　+　－　＊　/　][　またはスペースは含むことができません。 数字は文字と組み合わせてのみ使用できます。 解決策：  に文字を入力してください。
_____を実行することができません	ピコブロックスが値を返すブロックを実行しようとしましたが、 その値を受け取るブロックがついていませんでした。 解決策：その形状に適合する（および値を受け取る）ブロックを 添付してください。たとえば、次のようにします。

テキスト言語専用エラーメッセージ

以下のエラーメッセージは、ピコブロックステキスト言語を使用する場合にのみ表示されます。

エラーメッセージ	説明
ブロック _______ は4つ以上の 入力を持てませんa	4つ以上の入力 必要とするオリジナルブロックは作成できません。 解決策：他の値をグローバル変数として保管し、それらの変数の 名前をプログラムで使用してください。詳細については、この 取扱説明書のピコブロックステキスト言語セクションにある グローバル変数を参照してください。
______ は値を持っていません	ピコブロックスが、指定された名前の値を見つけられませんでした。 解決策：[テキストエリア]で、名前を正しく入力したか確認してください。 入力はその前にコロンを置きますが、グローバル変数の場合には 置きません。
出力ブロック _______ は 1つの入力しか持てません	出力するブロックには、1つの入力しか設定できません。 解決策：他の値をグローバル変数として保管し、 それらの変数をプログラムで使用してください。 詳細については、この取扱説明書のピコブロックステキスト言語 セクションにあるグローバル変数を参照してください。
() エラー)の場所が間違っています	丸括弧：()が欠落しているか、間違って配置されています。 解決策：開き括弧および閉じ括弧（ ）の数が等しいか、 そしてそれが正しく配置されているか確認してください。

エラーメッセージ	説明
変数に ________ という名前は つけられません	グローバル変数に別の名前を入力する必要があります。 解決策:グローバル変数の名前には、文字と数字だけを使用して ください。詳細については、この取扱説明書のピコブロックス テキスト言語セクションにあるグローバル変数を参照してください。
______は入力名として使えません	入力のあるオリジナルブロックを作成する際に、名前には 数値を含めることはできません。 解決策:別の名前を入力してください。 この際、数値は使用しないでください。 また、名前の前には、必ずコロンを置いてください。 例: block mydance :turns repeat :turns [motoronfor 10 reverse] end
[がありません] がありません	角括弧 :] または [が欠落しています。 解決策:開き角括弧:[および閉じ角括弧:]の数があっているか 確認してください。
to の後に名前がありません block の後に名前がありません bblock の後に名前がありません	解決策:to、blockおよびbblockは、その後に 任意の名前を 置く必要があります。たとえば、次のように記述してください。 block mylights setlightcolor 60 wait 10 lighto? end
______ が出力できません	指定した文字列またはブロックで適切な出力が行われません。 解決策:出力を必要とするすべてのブロックに出力用の ブロックが添付されているか、確認します。出力プロシージャ の使い方については、ピコブロックステキスト言語セクション を参照してください。
変数が多すぎます	プロジェクト内のグローバル変数が多すぎます。変数の最大数 (box1およびbox2を含めて)は7つです。 解決策:プロジェクト内で合計7つ以下になるように、 グローバル変数をいくつか削除してください。
waituntil は入力として リストが必要です	waituntilには入力が必要です。 解決策:waituntilの後にある角括弧:[ ]入力を行う コマンドを入力してください。 例: waituntil [touch?] chirp

エラーメッセージ	説明
ローカル変数が必要です	makeは、その後にローカル変数の名前を置く必要があります。 解決策:makeに続く 文字の前に引用符があり、 ローカル変数であるかを確認してください。 例: block fastlight :n setlightcolor :n make “n :n / 10 wait :n lighto? end